

大数据搜索引擎 原理分析及编程实现

刘凡平 编著

电子工业出版社

Publishing House of Electronics Industry

北京·BEIJING

内 容 简 介

本书向读者提供了一套完整的大数据时代背景下的搜索引擎解决方案，详尽地介绍了搜索引擎的技术架构、算法体系及取得的效果，以模块化的方式进行组织。着重介绍了机器学习在搜索引擎中的应用，包括中文分词、聚类、分类等核心的机器学习算法，并结合示例加以介绍和分析，使读者可以更好地理解机器学习在搜索引擎中的价值。还阐述了大数据给搜索引擎带来的新特性，结合目前大数据分析的主流工具，在搜索引擎中构建知识图谱，以及进行日志反馈学习机制，使得搜索引擎更加智能。

本书适合作为互联网行业从业者的技术参考书，也适合作为搜索引擎爱好者的参考读物。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有，侵权必究。

图书在版编目（CIP）数据

大数据搜索引擎原理分析及编程实现 / 刘凡平编著. — 北京：电子工业出版社，2016.7

ISBN 978-7-121-29164-7

I. ①大… II. ①刘… III. ①搜索引擎—程序设计 IV. ①TP391.3

中国版本图书馆CIP数据核字（2016）第141781号

策划编辑：李 冰

责任编辑：李 冰

特约编辑：田学清 罗树利

印 刷：

装 订：

出版发行：电子工业出版社

北京市海淀区万寿路173信箱 邮编：100036

开 本：720×1000 1/16 印张：20.5 字数：525千字

版 次：2016年7月第1版

印 次：2016年7月第1次印刷

定 价：59.00元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

本书咨询联系方式：libing@phei.com.cn。

前言

搜索引擎本身作为一门综合性的互联网技术，在行业中一直具备较高的关注度。随着最近几年大数据的发展，搜索引擎的关注度越来越高，原因在于搜索引擎技术是大数据应用最前线的领域，也是最容易产生价值的大数据应用。大数据存储、大数据计算都是从搜索引擎中衍生出的新领域。目前搜索引擎技术的发展不仅以大数据为基础，还利用分布式实时计算对数据进行高性能处理，以及利用机器学习将数据变得更具价值。在行业中吸引了包括搜索研发工程师、算法研发工程师、大数据分析工程师、自然语言处理工程师、计算平台架构师、数据挖掘工程师等行业各类精英的关注，这些工程师占据了整个互联网研发体系的 50%~60%，在 BAT 中，甚至超过 60% 的是研发人员。

这类群体薪资水平处于互联网研发人员中较高水平，以猎聘网公布的数据显示，北京地区搜索引擎研发工程师年薪为 35 万~60 万元，大数据工程师年薪为 20 万~25 万元，大数据架构师年薪为 40 万~70 万元，等等。也正是由于薪资水平处于较高位，很多互联网相关从业者也积极关注大数据搜索引擎领域动态。

本书通过介绍大数据下的自然语言处理框架、大数据存储引擎、搜索引擎的分布式实时计算、高性能可扩展爬虫，以及利用大数据构建知识图谱、基于大数据日志的搜索引擎反馈学习等相关信息，不仅使读者对当代搜索引擎研发体系有一定的认识，还可以使读者在搜索引擎领域及大数据领域进行深入思考。

本书特色

本书以当前搜索引擎主流技术为基础，密切结合前沿技术发展趋势，行文



通俗易懂，由初步的原理性了解到各模块应用示例，并结合分布式存储、实时计算等，向读者提供了一套完整的大数据时代背景下人工智能搜索引擎的解决方案。

（1）内容循序渐进、行文有条不紊地介绍搜索引擎知识。

本书充分考虑了不同层次的读者对搜索引擎的理解程度，因此本书由简入深、独特的技术写作视角符合广大读者对于技术类读物的理解需求，使得读者能够在掌握搜索引擎基础的情况下，不断按照搜索引擎的设计深入理解。

（2）技术前瞻性强，注重最新主流技术在现代搜索引擎中的应用。

本书充分利用了最新技术发展的应用成果，在自然语言处理的基础上不仅结合大数据分析，还包括分布式计算、机器学习、知识图谱等当前大数据应用与分析处理的主流技术，摒弃了传统过时的研发体系及算法。本书中相关研发成果在当前甚至在未来 3~5 年，都具有实际意义。

（3）将技术理论与应用范例结合，具备较高的商业实用价值。

本书内容紧密结合当前一线工程师工作研究成果，将众多的技术理论以实际工作经验的方式展示应用效果。本书介绍的内容也广泛结合工作中的应用示例，并以搜索引擎工程实践的脉络流程介绍技术要点，使读者在短时间内能够掌握当前搜索引擎研发的技术理论。

本书结构

本书按照由浅入深、循序渐进的顺序对现代搜索引擎原理和实现进行介绍。全书分为 10 章，各章的主要内容如下。

第 1 章针对搜索引擎发展的过去、现在、未来的相关概要介绍，以及现代搜索引擎与大数据、人工智能的相互关系，使广大读者能够在了解现代搜索引擎的背景之下，去了解本书的后续内容。

第2章是对搜索引擎原理与技术的初步分析，从模块方面大致介绍爬虫、索引、缓存等；从技术方面大致介绍自然语言处理、知识图谱技术、海量数据存储、分布式计算等。目的是使得读者对搜索引擎的体系结构、部分技术有一定认识，便于读者深入了解后续章节。

第3章从自然语言角度开始深入分析原理和实现，自然语言是搜索引擎进行文本处理的基础，其中包括分词、词性分析、语义分析、关键词抽取、核心句抽取、聚类分类等。读者将会从本章中获得当前主流的自然语言处理技术相关知识。

第4章主要是针对大数据存储引擎的介绍。大数据存储是搜索引擎最先遇到的问题，解决数据存储问题可以使搜索引擎在数据分析、索引构建、知识图谱等工作持续进行。读者在本章会了解到大数据存储引擎的架构体系、数据模型、数据压缩、负载均衡等。

第5章介绍了分布式实时计算。由于搜索引擎处理的是海量数据，数据分析必须依靠具有较强数据处理能力的计算平台，因此搜索引擎通过分布式实时计算去处理大数据并在尽可能短的时间内返回处理结果。本章中，读者会了解到分布式实时计算设计架构、负载均衡及通信设计等相关知识。

第6章对爬虫进行了深入分析。读者在本章中将会深入理解分布式可扩展爬虫的体系架构，以及对网页如何进行解析，并抽取出结构化的数据信息。本章还涉及链接去重、网页去重、广告识别等相关算法原理。

第7章详细介绍了知识图谱构建。知识图谱是智能化搜索引擎重要的组成部分，利用大数据分析构建出较为合理的知识图谱信息是当前主流的方式。读者将会从本章中深入了解到知识图谱的详细构建过程，以及利用机器学习原理对知识图谱中的实体抽取、关系抽取等相关技术进行。

第8章详细分析了索引构建机制。索引的设计与构造是搜索引擎能够进行



快速检索的核心要件，主要针对文件检索的倒排索引与用于智能提示的字典树索引。本章不仅对倒排索引做了深入分析，对倒排索引的压缩、分布式存储等也做了详细介绍。

第9章深入分析了搜索引擎的整个对外服务工作流程。包括大数据分布式缓存、搜索智能提示、个性化搜索、图片搜索、搜索与广告等。读者通过本章可以详细了解到文本纠错算法、动态摘要算法、网页排序算法及搜索引擎的评价体系。

第10章探讨和分析了搜索引擎日志与搜索引擎本身的关系。搜索引擎日志记录了用户与搜索系统交互的整个流程。通过日志挖掘，不仅可以发现用户的自有特征和行为规律，还可以有效地帮助搜索引擎提升性能和效果。日志作为搜索引擎的核心数据之一，一直使搜索引擎技术中的各类算法不断向前发展。读者通过本章将学会通过搜索引擎日志分析用户特征、用户的部分搜索意图等相关知识。

读者对象

- 适合对自然语言处理及机器学习应用领域有兴趣的读者。
- 适合对现代搜索引擎相关算法有兴趣的读者。
- 适合对大数据分析、数据挖掘应用有兴趣的读者。
- 适合互联网行业的不同层次从业者。
- 适合从事搜索引擎优化的网络营销读者。
- 适合高校中学习计算机、软件工程等相关专业的读者。

目 录

第 1 章 引论	1
1.1 搜索引擎的过去	1
1.2 搜索引擎的现在	2
1.3 搜索引擎的未来	4
1.4 大数据与搜索引擎	6
1.4.1 搜索价值提升	6
1.4.2 用户价值提升	7
1.5 大数据与人工智能	7
1.5.1 人工智能发展	7
1.5.2 人工智能技术	9
1.6 本章小结	11
第 2 章 搜索引擎原理与技术	12
2.1 基本工作原理	12
2.2 基本模块结构	13
2.2.1 爬虫服务	14
2.2.2 索引服务	15
2.2.3 缓存服务	16
2.2.4 搜索服务	17



2.2.5	日志服务	19
2.3	技术概要	20
2.3.1	自然语言处理	20
2.3.2	知识图谱技术	21
2.3.3	海量数据存储	23
2.3.4	分布式计算	25
2.3.5	搜索排序技术	26
2.4	本章小结	27
第 3 章	自然语言处理框架	28
3.1	英文分词	28
3.2	中文分词	30
3.2.1	中文分词概述	30
3.2.2	基于词库的分词技术	31
3.2.3	基于条件随机场的中文分词	33
3.2.4	分词粒度	41
3.3	词性标注	41
3.3.1	隐马尔科夫模型概要	42
3.3.2	隐马尔科夫模型与词性标注	43
3.4	语义相似度	51
3.5	依存句法分析	53
3.5.1	依存句法分析概要	53
3.5.2	依存句法分析实现	56

3.6	情感倾向分析	59
3.7	文档关键词抽取	61
3.7.1	关键词抽取概述	61
3.7.2	基于 TF-IDF 算法	62
3.7.3	基于 TextRank 算法	64
3.8	文档句子相似度分析	67
3.8.1	句子相似度	68
3.8.2	文档相似度	70
3.9	文档核心句抽取	71
3.10	聚类分类	74
3.10.1	文本分类	75
3.10.2	文本聚类	80
3.11	语种检测	84
3.12	本章小结	87
第 4 章	构建大数据存储引擎	88
4.1	架构体系	89
4.1.1	结构概要	89
4.1.2	服务器上线	92
4.1.3	服务器下线	92
4.1.4	数据读取	93
4.2	数据模型	94
4.3	数据压缩	96



4.4	负载均衡	97
4.5	数据存储逻辑视图	100
4.6	本章小结	103
第 5 章	构建分布式实时计算	104
5.1	概述	104
5.2	设计架构	106
5.2.1	设计思想	106
5.2.2	基本框架	108
5.3	运行模式	110
5.4	负载均衡	111
5.5	通信设计	112
5.5.1	基本方式	113
5.5.2	分布式远程服务调用	113
5.6	容灾恢复	114
5.7	数据容错原理	115
5.8	数据处理设计示例	117
5.9	本章小结	118
第 6 章	分布式可扩展爬虫	119
6.1	爬虫体系架构	119
6.1.1	主从分布式结构爬虫	120
6.1.2	对等分布式结构爬虫	120

6.1.3	基于分布式计算平台爬虫	121
6.2	网页解析	122
6.2.1	状态码处理	123
6.2.2	链接去重	123
6.2.3	广告识别	125
6.2.4	网站地图	128
6.2.5	非网页数据获取	129
6.2.6	网页去重	130
6.2.7	链接提取	134
6.2.8	爬虫协议	135
6.3	网页结构化	137
6.3.1	网页的编码信息	137
6.3.2	网页的正文信息	138
6.3.3	网站的关键词信息	142
6.3.4	网站的标题	142
6.3.5	网页的发布时间	144
6.3.6	网站语言检测	144
6.3.7	其他结构化数据	145
6.4	网页抓取策略	146
6.5	爬虫权限应对	147
6.6	深网抓取	150
6.7	抓取更新策略	151
6.8	本章小结	153



第 7 章 大数据构建知识图谱	154
7.1 概述	154
7.2 搜索引擎与知识图谱	155
7.3 可靠数据源选择	157
7.4 实体抽取	158
7.5 关系抽取	159
7.5.1 关系抽取概述	160
7.5.2 隐藏关系抽取	161
7.5.3 结构化确定关系抽取	164
7.5.4 非结构化确定关系抽取	166
7.6 知识图谱检测	171
7.6.1 实体关系修正	171
7.6.2 实体对齐整合	172
7.6.3 实体歧义分析	174
7.7 知识推理与计算	175
7.7.1 知识推理	175
7.7.2 知识计算	176
7.8 知识聚类	179
7.9 智能搜索实现	181
7.9.1 模式匹配	181
7.9.2 知识拆解	182
7.9.3 合并求解	184

7.10 智能搜索扩展	186
7.10.1 常识性智能搜索	186
7.10.2 实时信息智能搜索	187
7.10.3 可交互式智能搜索	187
7.11 本章小结	189
第 8 章 索引构建机制	190
8.1 倒排索引	190
8.1.1 倒排索引概述	191
8.1.2 索引结构	192
8.1.3 构建过程	194
8.1.4 排序规则	195
8.1.5 索引压缩	196
8.1.6 更新策略	202
8.2 分布式存储	202
8.2.1 存储划分方式	203
8.2.2 存储平衡策略	204
8.3 存储索引	209
8.3.1 二叉搜索树	210
8.3.2 B 树	211
8.3.3 B ⁺ 树	213
8.3.4 B ⁺ 树与文件索引	214
8.4 字典树索引	216
8.4.1 字典树索引概述	217



8.4.2	字典树索引构建	219
8.4.3	字典树查询优化	221
8.5	本章小结	221
第 9 章	搜索服务构建	223
9.1	概述	223
9.1.1	体系结构	223
9.1.2	七何分析法	224
9.1.3	搜索语法	225
9.1.4	相关性排序	227
9.1.5	不安全信息过滤	231
9.2	大数据分布式缓存	235
9.2.1	缓存结构设计	235
9.2.2	缓存更新策略	236
9.3	文本纠错算法	237
9.3.1	中文文本纠错	237
9.3.2	英文文本纠错	241
9.4	结果显示算法	242
9.4.1	动态摘要	243
9.4.2	关键词高亮算法	246
9.4.3	网页快照	250
9.5	搜索智能提示	250
9.6	网页排序	254

9.6.1	基于 PageRank 的网页重要性评价	254
9.6.2	基于 Hits 算法的网页权威性评价	257
9.6.3	Hilltop 算法	259
9.6.4	网页作弊评价	260
9.6.5	网页排序调试	263
9.7	个性化搜索	264
9.7.1	个性化搜索示例	264
9.7.2	人工神经网络与个性化搜索	265
9.7.3	地理位置搜索	266
9.8	图片搜索	271
9.8.1	基于内容的图片搜索	271
9.8.2	基于文本的图片搜索	272
9.9	搜索与广告	274
9.9.1	广告投放策略	275
9.9.2	基于 User-Based 协同过滤的广告投放	275
9.9.3	基于 Item-Based 协调过滤的广告投放	277
9.9.4	基于混合模式广告投放	278
9.9.5	广告投放评价	279
9.10	搜索引擎评价	282
9.10.1	搜索评价概述	282
9.10.2	基于准确率、召回率及 F 值评价	283
9.10.3	归一化折扣累计增益	285
9.11	本章小结	288



第 10 章 基于用户日志的反馈学习	290
10.1 基于用户搜索词语的分析	290
10.1.1 发现搜索词的价值	291
10.1.2 发现不明意图下的用户行为	292
10.2 基于用户点击日志的分析	293
10.2.1 时间与搜索意图的关系	293
10.2.2 地理位置与搜索意图的关系	294
10.2.3 点击日志与同义词	296
10.2.4 点击日志与词语权重	297
10.2.5 点击日志与新词分类	298
10.2.6 点击日志与知识图谱	300
10.2.7 点击日志与网页重排序	301
10.2.8 点击日志与网页评价	303
10.3 基于用户的特征分析	304
10.3.1 用户跟踪	305
10.3.2 用户群体特征	306
10.3.3 用户个体特征	308
10.4 本章小结	309

第 1 章 引 论

每个时代都有属于每个时代的故事，每项技术的发展历程，总是与社会的发展紧密结合，似乎验证着技术服务社会这一理念。社会的发展刺激着各行各业技术的不断向前推进，尤其在互联网大数据时代，搜索引擎技术发展越来越快。

1.1 搜索引擎的过去

1990 年年初虽然万维网还没有诞生，但是搜索引擎的鼻祖已经诞生，它没有响亮的名字，但作为一个搜索工具，它已经可以发现各个 FTP 主机中的文件。再到后来属于搜索引擎的各个时代来临，包括信息的人工分类时代、相关性检索时代、网页链接价值检索时代。

(1) 人工分类时代。搜索引擎最初的样子有点像一个导航网站，里面对网站进行分类，通过目录导航。但是由于在当时社会中，网站数量本身就比较少，用户和搜索引擎之间没有请求交互，此时的搜索引擎仅仅作作为信息展示的载体。在人工分类时代，搜索引擎导航目录的整个构建过程都是通过人工完成的。



(2) 相关性检索时代。从这个时代开始，搜索引擎具备了和用户交互的能力。用户通过入口提交搜索请求。但是搜索引擎只考虑了用户搜索词与结果的相关程度，仅将文本相关度较高的结果返回给用户。虽然此刻的它依然比较简单，但却是现代搜索引擎的开始，或者说是搜索引擎历史上的一次重要跨越。

(3) 网页链接价值检索时代。这个时代的搜索引擎，已经开始分析网页链接给搜索引擎带来的价值，开启了搜索效果评估的新篇章。以 Google 为代表的搜索引擎公司在最初的网页链接价值检索中在搜索效果和商业价值上取得了巨大成功，引领着更多搜索引擎公司挖掘网页链接的深层次价值。

1.2 搜索引擎的现在

过去代表着历史，但历史终将是尘埃。随着互联网技术的不断发展，现在的搜索引擎已经是具备个性化、多样化、智能化、社会化的现代搜索引擎。

(1) 个性化。指根据用户特征进行定制化的搜索服务，核心在于发现和理解用户的搜索行为，以及理解隐藏的用户特征价值，从而辅助用户进行搜索，使得个性化搜索满足用户的搜索需求。

(2) 多样化。用户的一次搜索不再是仅仅局限于网页文档的搜索，更多的信息类型（如音乐、图片、视频等）也会根据用户的搜索需求适时展示。

(3) 智能化。现代搜索引擎基本特性之一，越来越多的搜索引擎从搜索结果的数据量上转换为对搜索结果精准智能化的重视。智能化搜索通过用户的交互，最大限度地了解用户意图，并给予最佳排序结果甚至最终答案。

(4) 社会化。在搜索引擎集中研究网页之间关系、链接之间关系的时候，社会化试图通过把用户加入，研究在搜索过程中人与人、人与信息之间的关系。

用户可以通过共享的方式将知识分享出来,以使得在搜索过程中结果更加准确,可通过问答、共享百科、圈子讨论等方式实现。社会化依赖于具体的社交平台,通过这些社交平台进行知识分享,然后搜索引擎将这些信息进行索引。社会化搜索是 Web 2.0 中诞生的产品,在 Web 2.0 诞生不久,各类百科网站、博客网站、问答网站、交友分享社区层出不穷,经过多年的发展,它们将互联网信息进行了一个高度的总结和扩展,社会化搜索是对搜索引擎利用 Web 2.0 中的互联网产品对搜索服务和体验的一次改进。随着移动互联网的兴盛,社会化搜索已经更加明显。

现在,无论是兴盛的移动搜索还是被视为日渐衰落的传统 PC 搜索,从技术角度看,它们的本质并没有太多改变。当前的搜索引擎已经发展出全网搜索引擎、垂直搜索引擎、元搜索引擎三大重要分支。

(1) 全网搜索引擎。网民使用最多的都是全网搜索引擎,当前几乎成为整个互联网的入口,面向所有网民使用。当前在全网搜索引擎领域各大厂家竞争激烈,如百度、搜狗、好搜(奇虎)等互联网公司。

(2) 垂直搜索引擎。垂直搜索引擎的数据限于特定垂直领域。垂直领域是针对某一个行业或者细分领域的,这些垂直领域是全网搜索引擎的子集。垂直搜索引擎的采集数据源具备针对性,面向特定领域或特定人群使用,如学术、图片、视频搜索等。

(3) 元搜索引擎。是在用户输入搜索词之后,根据其他多个搜索引擎合理组织出新的数据,从而返回组织后的结果,元搜索引擎没有自己的爬虫。

当前的搜索引擎的输入方式也发生了重要变化,不仅可以通过文本输入,还可以通过图片、语音等输入。尤其语音输入,是当代搜索引擎在移动互联网发展下的重要变化。随着语音识别技术的发展,给语音搜索发展创造了机会,从而产生了更加便捷的搜索方式,尤其是在移动搜索方面。



知识图谱是通过对常识、领域知识等建立的一种关系图结构。知识是对信息总结的描述，搜索引擎之所以会逐步发展到知识图谱，是因为世界并不是由文字组成的，而是通过各个实体之间的相互作用而形成的，因此搜索引擎从研究文本本身，转向于研究自然界知识实体之间的关系。每一个知识实体或者概念知识都在图谱中拥有全局唯一的标识。知识图谱目前主要在互联网大数据领域、图书情报领域（如引用分析、思维导图、复杂社会网络等领域）的发展比较迅速，这些领域基本上是以互联网大数据领域发展为基础。如果未来的搜索引擎是机器人的大脑，那么知识图谱则是这个大脑的知识库，任何决策都依赖于此知识库。当前各大互联网公司已经在大力发展知识图谱相关项目，可谓战略意义重大。

在搜索引擎使用知识图谱之前，使用的是名为知识卡片的中间产品，但是很快就成功过渡到知识图谱。从知识图谱理解知识卡片即每一个知识图谱上的实体，都拥有它自己的描述信息。

当前搜索引擎的数据除通过分布式爬虫获取之外，还通过开放平台使得数据更具实时性和有效性，例如，针对火车票、飞机票等的搜索，都是基于搜索引擎开放平台。

此外，在大数据和云计算的发展推动下，更多的互联网厂商也逐步涉足搜索引擎垂直搜索领域，如电商、房地产等领域。这些垂直搜索引擎与丰富的商业模式结合，使得搜索引擎更加专业和智能化，也正因如此，搜索引擎与传统领域相互结合，成为了“互联网+”的重要体现之一。

1.3 搜索引擎的未来

在过去 30 年，互联网 Web 技术从 Web 1.0 进化到 Web 3.0。Web 1.0 打开了人与网络，网民可以自由地浏览互联网信息。Web 2.0 又称为社交网络（Social

Web), 产生了大量以社交为主的服务网站, 如开心网、微博、人人网。当前的 Web 3.0 更是以大数据和智能为主题的时代, 不仅对搜索结果追求更加极致, 而且对于多媒体信息的搜索需求已经开始急剧增加。即将蓬勃发展和到来的 Web 4.0 将会是无处不在的网络, 搜索引擎将会在这个无处不在的网络中, 发挥其更加独特的作用。

从笔者角度来看, 未来的搜索引擎应当是“人工智能”化。当前的搜索引擎在个性化、智能化、垂直化、简单化、社交化等方面都已经比较成熟。这必将使得人们对信息的追求发生改变, 甚至希望搜索引擎成为每一个人的“智能伴侣”。因此, 笔者认为搜索引擎是开启未来人工智能世界的一把钥匙。

未来的输入方式也不再限于文本、语音、图片等输入, 还可能包括体温、温度、情感等各类生活环境的输入。输出方式也会变得多种多样, 具备更强的交互性, 如动态媒体输出、文字描述的动画展示理解等。

人类之所以比其他生物聪明, 在于人类善于“归纳、总结、学习”, 而这些都是当下搜索引擎试图做得更好的方法。搜索引擎将会从一个互联网工具转变为人工智能平台, 移动互联网、物联网、PC 互联网领域, 搜索引擎都可以与各大产业相互结合, 以帮助其解决问题。未来的机器人时代定会是以搜索引擎为基础的大脑控制系统, 目前来看其他一般系统则很难完成, 原因在于机器人和人一样都在随时学习, 而学习的必经途径就是搜索引擎, 通过搜索引擎发现信息、拓展知识、自我学习, 而机器人也会一样。俗话说“授人以鱼, 不如授人以渔”, 而从机器人角度看, 这个“渔”不正是未来的搜索引擎吗?

从现代社会来看, 年轻人经常“提笔忘字”, 他们似乎在失去部分记忆, 而且越基础的知识越容易失去。偶尔甚至连最基本的算术计算, 现在都使用计算器或者搜索引擎完成, 在未来我们的更多工作会交给搜索引擎, 人类的意识和思考将会更加依赖搜索引擎。



1.4 大数据与搜索引擎

搜索引擎一直在海量数据中挖掘最优质信息为网民提供最佳服务。在大数据时代，新的数据给予搜索引擎新的篇章。大数据的“大”决定了数据大而全面的特性，在传统计算模式中，通过抽样数据分析得到的结果已经不如通过大数据分析得出的结果精确。大数据挖掘几乎可以拿到任何想要的信息。搜索引擎经过最近十几年的发展，已经在文本分析、数据挖掘、图谱构造、语义分析等方面有了丰富的积累，结合现有大数据将会给现有搜索技术带来全方位提升。这些改变不仅可以帮助搜索引擎找到更加准确的答案，还有助于搜索引擎个性化精准分析。

1.4.1 搜索价值提升

现代的人类学习、生活、工作都已经离不开信息，从过去的信息大爆炸到如今的数据大爆炸，搜索引擎不仅能帮助用户从海量信息中找到结果，更是一种互联网服务。搜索引擎成为一个数据工厂，通过大数据挖掘，抽象结构化有价值的信息，加速信息流动，促使搜索为用户提供更多服务及更高价值。

让用户对搜索结果进行筛选的时代很快就会被抛弃在时代的浪潮中，当前大数据时代研究最多的深度学习也是研究搜索引擎能够直接命中用户答案的途径之一。现在的搜索主要是基于互联网中已经存在的数据，返回结果。但是在大数据背景下，搜索引擎可以发现互联网中不存在的信息。搜索引擎通过自我学习，给出一个参考，例如，用户搜索“下周股市会大涨吗？”在互联网中一定不会有“下周股市会大涨”的确切数据，但是会有很多相关性点评，很多业内人士对下周股市的看法，深度学习可以通过汇集行业相关人士看法、行业市场状况等信息，为网民提供可靠的参考信息。

大数据的发展使得搜索引擎成为更加开放的搜索平台，伴随着计算机视觉、深度学习等领域的发展，社会信息将会发生巨大变化，搜索引擎需要对外开放其大数据存储、数据分析能力、智能化处理等，吸引更多的开发者、企业，以弥补搜索引擎在外部资源中的不足，从而促使其不断提升搜索价值。

1.4.2 用户价值提升

用户在互联网中留下的任何数据，都已经成为搜索引擎研究的数据对象，搜索引擎通过基于海量数据的机器学习、统计学、模式识别等技术，从大数据中提取出其中隐藏的有效信息，逐步成为用户的知心朋友。

用户获得信息的方式不再仅仅是主动的搜索，还包含被动的信息接收。当用户在搜索过程中寻找相关信息时，搜索引擎通过大数据分析可以为用户提供解决问题的更多可能性，实现从主动寻找答案转变为被动接受答案。

1.5 大数据与人工智能

大数据使得人工智能技术开始新的发展，而人工智能技术也深入影响着搜索引擎。例如，传统搜索方式中，通过搜索尽可能显示全面的信息，而现在只需要显示满足用户需要的信息即可。

1.5.1 人工智能发展

大数据人工智能是建立在大数据基础之上的人工智能技术。人工智能（Artificial Intelligence，AI）是对人类的意识、思维及信息处理过程的模拟。曾经，很多人都说人工智能“已死”，一直处于停滞不前的状态，但是伴随着近几年大数据的发展，人工智能似乎“死而复生”，获得了新的发展机遇。而搜



搜索引擎利用人工智能技术的原因是希望其搜索结果更加精准，甚至直接命中答案。Google 也在不断利用人工智能技术改善它的搜索技术，还开源了其机器学习系统（代号：TensorFlow），相似机器学习系统还有 Facebook 开源的 Torch、微软开源的 DMTK、IBM 开源的 SystemML。

人工智能技术是计算机科学发展的重要分支。从 1956 年达特茅斯会议中诞生，人工智能技术经历了两次大起大落，它的第一次黄金年代是 1956—1974 年，这个阶段中计算机逐渐可以解决代数应用相关问题，LISP 语言也在 1960 年诞生，是令人兴奋的一个开端。但是后来对于人工智能的过度乐观，研究项目甚至也不能如期完成，研究者的项目资助也出现了问题，甚至连哲学家也开始反对人工智能。1974—1980 年是人工智能发展的低谷期。

从历史的角度分析，每一次失败，都在为下一次发展做铺垫，技术的发展似乎如过山车一般，后来的专家系统程序在人工智能领域发展势头较好，得到了不少企业的认可，使得从 1980 年开始，人工智能技术似乎迎来了新的繁荣。但是也再次验证期望和失望是正比例相关，好景不长，仅仅 7 年之后，人工智能的冬天就又来到。这将注定是好事多磨，直到 20 世纪 90 年代中期以后，人工神经网络研究取得新进展，以深蓝战胜国际象棋的世界冠军为里程碑，再加之计算机性能随着摩尔定律疯狂似的增长，似乎又进入了新的人工智能时代，尤其在当前的大数据和云计算时代，数据的多样性和丰富性更加显现，数学模型和算法也在不断地改进，人工智能仿佛迎来了它的第三次繁荣。

2016 年 3 月，Google 的人工智能围棋程序 AlphaGo 对战世界围棋冠军、职业九段选手李世石，并以 4:1 的总比分获胜，人工智能再次展现出蓬勃的生命力，未来充满了无限的想象。

在当前技术发展趋势下，人工智能（弱人工智能）已经不刻意去研究模拟人脑，而是利用大数据通过计算机视觉、数据挖掘、统计理论，深化演绎、推理、

归纳去解决现实中的各类问题。人工智能对互联网的发展和帮助是不言而喻的，如现实中的指纹识别、人脸识别、视网膜识别、专家系统等，都是人工智能的实际应用案例。

任何技术的发展之路都是崎岖的，人工智能则是典型的代表，从科学技术发展史来看，技术解决人类的需求是永无止境的，无论现在的人工智能发展到哪个水平，它对于搜索引擎的促进作用都是潜移默化的，两者在未来的发展中将更加紧密相连。机器学习作为现代搜索引擎的基本工具，在搜索引擎的多样性、个性化、准确性方面有着举足轻重的作用，成为当代搜索引擎智能化的标志。因此对待人工智能技术，以一种坦然和宽容的方式，它的进步焦点不在于它能否超过人类的智慧，而在于其是否帮助人类更好地生活。

1.5.2 人工智能技术

大数据人工智能技术在应用层面包括机器学习、人工神经网络、深度学习等，它们都是现代人工智能的核心技术。在大数据背景下，这些技术得到了质的提升。

(1) 机器学习。又称为统计学习理论，通过数据分析获得数据规律并将这些规律应用于预测或判定其他未知数据。机器学习目前已经广泛应用于数据挖掘、自然语言处理、语音识别等，尤其在搜索引擎领域，在海量数据面前机器学习的方法成效显著，具体算法包括决策树、感知器、支持向量机、马尔科夫链、最近邻居法等。

(2) 人工神经网络。是机器学习的一个重要算法，人工智能技术中，人工神经网络（Artificial Neural Network, ANN）是常用的方法之一，是一种通过模仿生物的神经网络结构和功能的数学模型，也是一种自适应的计算模型，通过感知外部信息的变化，改变系统内部结构。人工神经网络由许多神经元组成，神经元之间相互联系构成信息处理的庞大网络。假设做一件事情有多种途径，



那么神经网络会告知用户哪一种途径是最佳方式。人工神经网络的优势在于，它是一个能够通过现有数据进行自我学习、总结、归纳的系统，能够进行推理产生一个智能识别系统。

(3) 深度学习 (Deep Learning)。它是机器学习的重要分支，深度学习的模型框架已经很多，如深度神经网络、卷积神经网络、递归神经网络等，作为多层非线性神经网络模型，它拥有强大的学习能力，与大数据结合，再利用云计算、GPU 并行计算，使得其在图形图像、视觉、语音等方面获得较高评价，因此深度学习被大众视为人工智能前进的重要一步。

除此之外，人工智能的基本技术包括自然语言处理、知识表示与知识库技术、推理技术、搜索技术（问题求解）等。而这些技术也是当前搜索引擎的基础，结合大数据它将为搜索引擎带来潜移默化的作用。

(1) 自然语言处理。用户常常通过不断更换调整搜索词以获得更好的搜索结果，大数据时代背景下的搜索引擎更需要了解用户的搜索意图，以减少用户的搜索次数。而自然语言处理即是帮助搜索引擎理解用户意图的重要方法之一，自然语言处理不仅是对用户的搜索进行分词，还需要做到语义分析，甚至信息拆解。自然语言在搜索引擎中的处理过程不仅针对用户的搜索，还包括对网页文档的分析、表述观点分析等。虽然目前自然语言处理中的一些模块已经做得很好，但是在大数据背景下自然语言处理还有很大的提升空间。

(2) 知识表示与知识库技术。其主要的目标是存储知识，一个好的存储方式可以让程序更加灵活。涉及的问题包括“知识的本质是什么”“如何表示知识”“表达方式是否通用”等。在搜索引擎中，知识图谱作为现在知识表示和知识库技术的呈现，虽然研究者不确定知识图谱是否是知识表示和知识库技术的最终表现形式，但至少知识图谱已经为搜索引擎带来了巨大的改变。

(3) 推理技术。推理作为人类思维中解决问题的主要思考模式，主要利用

知识及知识之间的关系，获得推导信息的过程与结论。推理包括演绎推理、归纳推理和默认推理。例如，已知明明的爸爸是大明，而大明的爸爸是老明，根据已知条件，能够推理和判定得出明明的爷爷是老明。

（4）搜索技术。搜索技术不是搜索引擎技术的简称，而是人工智能领域中的专业术语，实质是一个问题求解的过程，简单的如“八皇后问题”“九宫图”“河内塔”等，在实际算法中，类似包括“A*”算法，在传统算法或游戏中，搜索技术在寻求一个问题解决的最佳办法，搜索引擎也一样，在给用户提供最佳答案。

1.6 本章小结

搜索引擎从历史中走来，向未来迈步跨进，在不足 30 年的时间里，取得了高速发展。但目前的搜索引擎依然不能完全满足用户需求，未来的搜索引擎将会朝着更加智能，并结合知识与情景交互的方向发展，连接人与服务，为用户提供更加准确的搜索结果和最适合用户的数据。在大数据背景下的互联网时代，搜索引擎将是改变人类生活的引擎。搜索引擎通过大数据人工智能技术不断提升其“智商”，逐步成为每一个人的“伴侣”与信息专家，但是无论怎样发展，它们都需要技术的强大支持。本章通过对搜索引擎历史的介绍及大数据、人工智能的相关分析，使读者对搜索引擎能够产生初步的了解。

第 2 章 搜索引擎原理与技术

互联网的数据每天都在不间断增长，搜索引擎对日新月异的互联网数据进行抓取获取，并经过数据整理之后，为用户提供搜索服务，而这些过程都需要较好的工作流程和技术体系作为支撑。理解搜索引擎的工作原理和基本技术是认识搜索引擎的基础。

2.1 基本工作原理

搜索引擎技术不同于传统的全文检索技术，虽然搜索引擎技术是基于全文检索的技术，但是也存在不同点，包括对数据的处理量、处理性能、体系结构等方面。

（1）数据的处理量。搜索引擎技术面向的是互联网海量数据整合，并提供检索服务，而全文检索服务针对的是小规模数据，例如对企业内部数据进行的检索服务。

（2）处理性能。搜索引擎技术不仅仅需要快速获得互联网信息，还需要在最短的时间内反馈用户的请求。而全文检索技术不仅大多数据已经存在，而且数据差异性较小。全文检索技术仅是对数据进行全文索引，对检索时间性能要求也没有搜索引擎高。

(3) 体系结构。搜索引擎是一套完整的技术体系,包括爬虫服务、索引服务、缓存服务、搜索服务、日志服务等一系列技术,而全文检索更多针对于索引服务与搜索服务。

正是由于上述不同点,从工程应用角度看,搜索引擎技术的难度远远大于全文检索技术。对于搜索引擎的工作原理,简单地说,搜索引擎后台首先进行互联网信息采集,建立结构化网页数据库;然后对数据建立索引并构建索引库;在用户访问搜索服务器之后,先通过缓存服务器获得可能缓存的搜索数据,若缓存服务器中未命中相关数据,则通过后台建立的索引查询出与用户搜索相关的网页,最后利用网页存储数据库在搜索结果中显示网页标题及部分内容摘要等信息。图 2-1 所示为搜索引擎工作原理的简单结构示意。

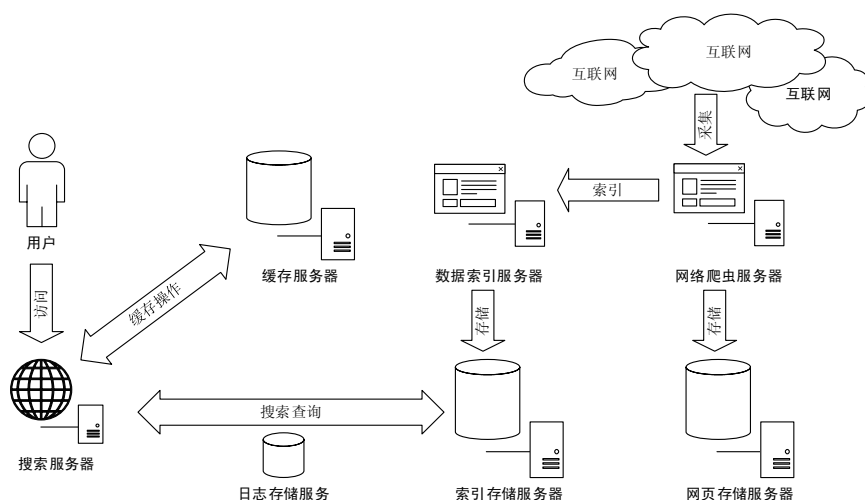


图 2-1 搜索引擎工作原理的简单结构示意

2.2 基本模块结构

在搜索引擎的结构体系中,主要包含网络爬虫服务(Web Crawler)、索引

服务、缓存服务、搜索服务、日志服务等几大服务模块。各服务模块之间相互影响，构成了搜索引擎运作的整个流程。

2.2.1 爬虫服务

网络爬虫又称为网络蜘蛛（Web Spider），是一种基于互联网的自动化浏览程序。互联网通过网页链接相互之间产生关联，形成巨大的网络图结构。网页链接一般简称为 URL，实质上是互联网中的统一资源定位符，是互联网资源存放位置的标准地址。因此，爬虫作为搜索引擎对数据获取的工具，通过 URL 对互联网进行尽可能广泛的遍历，使得搜索引擎拥有海量的互联网信息。普通爬虫的基本功能逻辑如图 2-2 所示。

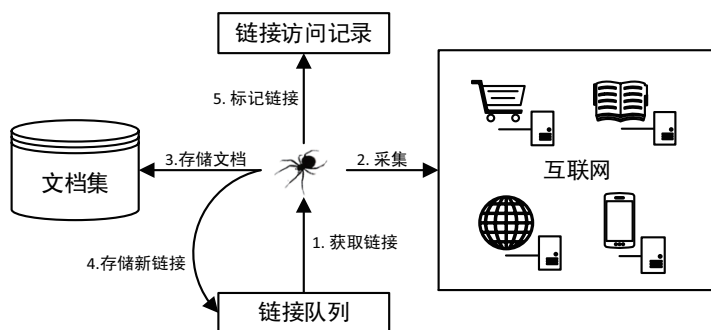


图 2-2 普通爬虫的基本功能逻辑

（1）爬虫从已经初始化好的网页链接队列中取出种子链接（如 <http://www.iveely.com> 等），通过这些种子链接不断地从互联网中获得新的网页数据。

（2）通过网页链接下载相应网页数据，通过分析网页数据提取新的链接存储到链接的后续队列中，且将访问过的网页链接进行已访问标记。

（3）依次不间断地从队列中获取链接并逐一访问，理论上链接集合中的所有链接均被访问后，爬虫将停止工作。

在爬虫不断获取新数据的过程中，也会定期更新一部分网页，使得数据尽可能保持时效性。

由于互联网数据不仅量大，而且更新频率也较快，然而一般爬虫每天下载的互联网资源有限，这意味着爬虫的设计者必须对爬虫架构进行优化，并且需要对资源下载方式及下载性能进行全面考虑，以保证在最短时间内下载更多的互联网数据信息。除此之外，还需要考虑互联网资源的持续可访问性，并对网页去重、链接去重等一系列问题深入思考。

2.2.2 索引服务

索引是一种用于数据快速查找的数据结构，例如，本书的目录也是一种索引结构。索引的价值在于在最短的时间内获得最相关、最全、最深的信息集合。

对于搜索引擎而言，要在数十亿的网页中筛选出最合适的网页并不容易，虽然用户通过搜索词搜索互联网资源，若想通过依次遍历的方式获得最相关的网页，从工程应用角度来说则不具备可行性，主要是由于时间性能和资源访问的效率不允许。在对爬虫获取的网页进行分析的过程中，可以发现互联网上的网页与词语是一个巨大的矩阵，如表 2-1 所示。

表 2-1 网页与词语的对应关系矩阵

	网页 1	网页 2	网页 3	网页 4	网页 5
词语 A	√	×	×	√	×
词语 B	×	√	×	×	√
词语 C	√	×	√	×	√
词语 D	×	√	×	√	×
词语 E	√	×	×	×	√

表 2-1 中，“×”表示该网页不包含对应词语，“√”表示网页包含对应的词语，例如，对于网页 1 则包含词语 A、词语 C、词语 E，不包含词语 B、词语 D。如果用户在搜索词语 B 时，对网页 1~5 依次遍历，查找包含词语 B 的网页，

显然不是一种可取之法，但是可以通过表 2-1 得知，与其从矩阵的纵向去寻找包含词语 B 的网页，不如横向比较词语 B 在哪些网页中出现过。这个横向比较的结构，即为搜索引擎索引结构中典型的倒排索引（Inverted Index）结构。

因此，为保证用户对互联网数据的快速有效访问，索引服务对爬虫抓取到的数据建立倒排索引。倒排索引是搜索引擎中核心的数据结构之一，尤其是针对海量数据的索引。倒排索引不仅实现简单，操作也极为方便。倒排索引的合理使用可以使得搜索性能、搜索相关性都能达到较为满意的期望。

索引拥有了良好的数据结构是作为快速访问的基础，面对庞大的网页数据，对其建立索引，还需要工程架构上的支持，采用分布式索引分析、索引存储等可以使得建立索引的时间和索引访问的时间事半功倍。当然为了保证良好的搜索体验，索引的更新也是开发者面临的巨大挑战。

2.2.3 缓存服务

缓存服务在一般大型的分布式应用中都会涉及，目的在于减少磁盘 I/O 及网络带宽等资源消耗。搜索引擎作为典型的分布式应用，也不例外利用缓存服务对外提供更加优质的搜索体验。缓存服务在一般的分布式应用中的角色如图 2-3 所示。

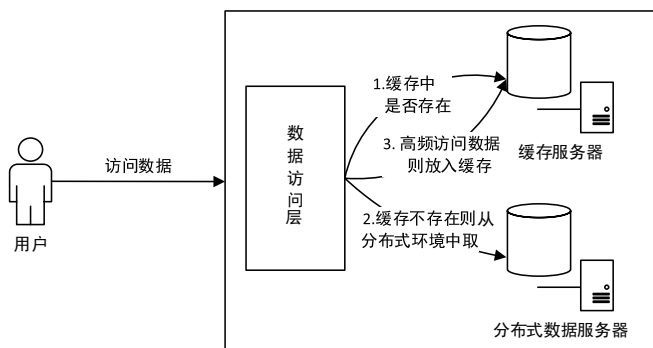


图 2-3 缓存服务在一般的分布式应用中的角色

通过图 2-3 可以知道用户通过数据访问层进行数据访问时，数据访问层会通过缓存服务器去检查用户访问的数据是否在缓存中存在；如果不存在则会通过数据服务器进行数据获取操作；数据访问层检测到用户高频率访问某一数据时，会将该数据存入缓存服务器，以便用户下次快速获取。

同一般的缓存服务器一样，搜索引擎的缓存是搜索引擎对外服务过程中帮助用户快速进行数据访问的策略之一，目前几乎所有的搜索引擎都会采用缓存技术。搜索引擎的缓存机制是在用户的大量搜索词中，选择搜索流量较高的搜索词，将其放入缓存服务器，在下次用户搜索相同搜索词时，从缓存服务器中直接获取数据，而不再访问数据后台。搜索引擎引入缓存服务的目的的一方面是加快用户查询的响应速度；另一方面是减少服务器后台计算量，节省系统资源，以使得后台资源更加高效利用。

搜索引擎的缓存服务器不同于其他中小型缓存服务，主要搜索引擎的应用场景使得缓存对分布式要求及实效性要求更高。

(1) 分布式协同缓存。搜索引擎缓存的搜索信息不仅复杂多变，而且量非常庞大，采用简单的几台服务器做缓存服务，很难满足搜索引擎的缓存需求，间接性会导致后台请求压力过大等情况。所以搜索引擎需要采用分布式协同缓存的方式，并且对此分布式技术架构也提出了新的要求，需要保证稳定的提供缓存服务，否则不仅会导致用户搜索体验下降，更会导致搜索引擎流量安全。

(2) 时效性要求更高。实效要求不仅是指缓存服务器能够快速反馈结果信息，还要保证返回缓存结果具备时效性，也就是对缓存更新策略提出了更高的要求，例如，将一个月前缓存的搜索信息保存到今天是搜索引擎不能允许的。

2.2.4 搜索服务

搜索服务包括搜索服务接口及搜索服务后台两方面。其中，搜索服务接口

是用户与搜索引擎进行交互访问的接口，目前主流的搜索服务接口包括网页站点、移动应用和桌面助手。

(1) 网页站点。是基于传统的方式采用 HTML 搭建的网站，用户通过访问网站进行搜索获得搜索结果，如百度的 <http://www.baidu.com>、好搜的 <http://www.haosou.com> 等。

(2) 移动应用。在移动互联网蓬勃发展的今天，各大搜索引擎公司均提供了基于移动操作系统的移动应用，提供搜索服务，如手机百度、搜狗搜索客户端等。

(3) 桌面助手。这是在智能时代一种新的搜索方式，如微软公司的桌面应用 Cortana、百度公司的百度桌面应用等，都是基于桌面操作系统平台构建的应用软件。

但是无论采用上述哪种搜索服务接口，它们对应的搜索服务后台是一致的，搜索服务后台是将用户提交的搜索关键词获得返回相应数据信息结果，并对搜索结果的数据进行整理的过程，包括排序、个性化处理等。缓存服务也是在搜索服务接口访问搜索服务后台过程中产生作用，如图 2-4 所示。



图 2-4 搜索接口与搜索后台对应关系

搜索服务作为搜索引擎最重要的组成部分之一，用户对其非常敏感，不同于爬虫或者缓存服务，从用户角度看，不管爬虫多么强大、缓存多么高效，如果搜索服务不能达到预期，用户就会对该搜索引擎产生负面心理。一个具备极限用户体验的搜索服务接口和极致智能的搜索服务后台是一个商业搜索引擎的基础，目前大多数搜索引擎公司在搜索交互及搜索服务后台智能化方面不懈努力。

2.2.5 日志服务

日志并不是搜索引擎内部主动产生的数据，属于搜索引擎的外部数据。用户在搜索引擎上每天产生大量的搜索记录，这些搜索记录包括搜索词与搜索词之间的关系、搜索词与用户之间的关系、搜索词与点击情况的关系等，而所有这些情况称为搜索日志。

日志服务是对用户产生的搜索日志进行进一步分析，为提供更好的搜索体验及搜索引擎的技术自我升级提供保障。因为没有任何一个搜索引擎在刚诞生的时候就是完美的，更何况互联网数据日新月异的变化。日志服务对于实事焦点信息分析、增进对用户的了解、完善搜索技术三个方面拥有较好的帮助。

(1) 实事焦点信息分析。通过日志可以知道在某些事件范围内，网民最关心的事件焦点、最热门的电视剧、最爱玩的游戏，甚至还可以分析出最新出现的网络新词、网络热词等。

(2) 增进对用户的了解。用户不间断地通过搜索引擎产生搜索行为，他们的行为点点滴滴都被记录，搜索引擎通过分析用户的行为，不断增加对用户的了解，为用户推荐更好的搜索结果。

(3) 完善搜索技术。排序是搜索引擎中非常重要的组成部分，但是搜索引擎在最初很难发现当前的排序是否能够满足用户的需求，因此通过用户的日志分析，可以计算用户在使用相关搜索词时，对返回的排序结果是否达到了期望，如果没有达到期望，则不断反馈学习调试，最终使搜索技术不断完善。

在大数据时代，需要对日志进行分析并且达到了解用户、搜索技术自我完善的目的，仅仅是普通的数据分析，还不能满足。需要通过一定的分析模型及大数据处理平台，才能更加有效地分析出准确结果。

2.3 技术概要

搜索引擎中各大服务模块构成了搜索引擎的工作流程，但是每个服务模块需要正常运作，还需要强大的技术支撑，这些技术主要包含自然语言处理、知识、图谱海量数据存储、分布式计算及搜索排序等，其中的每项技术都会对搜索引擎的搜索结果产生非常重要的影响。

2.3.1 自然语言处理

自然语言处理（Natural Language Processing，NLP）是人工智能领域与语言学的交叉学科，目的在于让计算机能够认知人类语言。在搜索引擎技术中自然语言处理主要用于海量数据的文本挖掘。

在搜索引擎获得海量互联网数据之后，利用大数据分析原理及自然语言处理对数据进行文本挖掘，以发现更多具有价值的信息。文本挖掘主要分为文本分析及特征分析。文本分析包括分词技术、词性分析、语义分析、依存句法分析和句子相似度等。特征分析包括文本语种检测、文档核心句子提取、文本关键词提取、文本情感分析以及文本聚类 and 分类等。文本挖掘流程如图 2-5 所示。

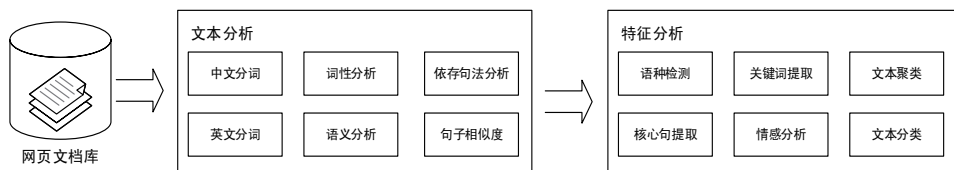


图 2-5 文本挖掘流程

文本挖掘中采用的各大自然语言处理模块的具体含义如表 2-2 所示。

表 2-2 文本挖掘中采用的各大自然语言处理模块的具体含义

自然语言处理子模块	含 义
中文分词	对中文句子进行分词处理，将完成一句话处理为词语的集合
英文分词	将英文进行词形还原，并将英文句子拆分为一个一个单词的形式
词性分析	主要针对中文分词后的词语，确定词语在句子中的词性成分，如名词、动词等
语义分析	针对不同的词语，计算在语义上的相似度，例如，两个词虽然文字不同，但可能表述相同的信息
依存句法分析	将分词后的句子，对句子成分进行结构化分析，以及句中词语之间的相互关系分析
句子相似度	根据句子中的成分，确定句子在表述上的相似程度
语种检测	对处理文本的表述语种进行检测，并确定其所属语言，如简体中文、繁体中文、英文等
关键词提取	对处理的文本进行分析，抽取其最具典型代表的词语
核心句提取	对于一篇文档内容，由多句文本组成，选取最具代表该文档的句子
情感分析	获得文本在进行信息描述时，带有一定的情感，例如积极与消极程度
文本聚类	不同的文本，可能表述的是同一个类别的信息，将此类型的文本放置在一起
文本分类	不同的文本可能属于各种类别，将文本所属的类别进行识别，并给定其确定性类别

在搜索引擎的基本模块结构中，爬虫服务从互联网中采集的大量数据信息，在通过自然语言处理框架处理之后，将文本信息逐步分析成为结构化数据及价值性数据，实现构建知识图谱及数据索引。在搜索服务中也会使用自然语言处理去深入理解用户的搜索意图。

2.3.2 知识图谱技术

知识图谱目前已经是现代搜索引擎的标配，但是在大数据时代之前，很多公司还不愿涉足此领域，原因在于研发成本较高，但是现在已经截然不同。知识图谱技术是为搜索引擎提供精准答案的技术基础，目前几乎所有的搜索引擎公司都非常重视知识图谱的研发工作。知识图谱作为一种图结构，实质是对一

个结构化数据的形式化表达，它的构建是通过大数据不断分析各类知识信息、知识之间的关系最终构成的。

知识图谱中有三个非常重要的概念：实体、实体标签和实体关系，它们是知识图谱中基本的组成元素。

(1) 实体。表示具有分析价值的具体对象，包括人、物、时间、地理位置等，如“马云”“打印机”“1949 年”“北京”“颐和园”等。

(2) 实体标签。是在实体中，具有标识意义的标签信息，能够对实体做到一定的区分度。例如“中国艺人梁朝伟”，其中“梁朝伟”作为人名，被视为实体，“中国艺人”即实体“梁朝伟”的实体标签。

(3) 实体关系。顾名思义是指实体与实体之间存在的关系，例如“北京是中国的首都”，其中“北京”和“中国”分别代表两个实体，而“首都”表示两者之间的关系，即实体关系。

再通过具体的例子分析实体、实体关系，如表 2-3 所示，对“唐朝”与“李白”相关的两个句子进行实体与实体关系分析。

表 2-3 对句子进行实体与实体关系分析

句 子	实体 A	实体 B	实体关系
唐朝出生的李白出生于公元 701 年，逝世于公元 762 年	李白	唐朝 公元 701 年 公元 762 年	(李白) 出生朝代 (唐朝) (李白) 出生 (公元 701 年) (李白) 逝世 (公元 762 年)
唐朝的首都位于长安，使用方孔钱作为货币，开国皇帝是李渊	唐朝	长安 方孔钱 李渊	(唐朝) 首都 (长安) (唐朝) 货币 (方孔钱) (唐朝) 开国皇帝 (李渊)

上述完成了对实体及实体关系分析，但是还需要通过上述信息构建知识图谱，图 2-6 所示为依据表 2-3 构建的知识图谱信息，充分表现出实体、实体标签和实体关系相关元素。

对于实现图 2-6 中所示效果，默认情况下的句子中不会告知实体标签信息，

需要根据实际情况获取。完成整个知识图谱构建不仅依赖于自然语言处理技术,而且文本中的实体识别、实体关系抽取及实体标签自动化标注、实体对齐去重等,都是构建过程中必须解决的问题。

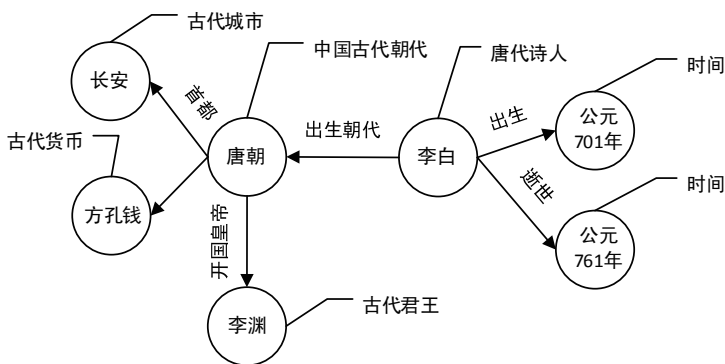


图 2-6 实体“唐朝”与“李白”知识图谱构建示例

2.3.3 海量数据存储

现代的搜索引擎在数据层面已经发生巨大的变化，无论是传统的 PC 互联网还是移动互联网，每天都在产生新的数据。搜索引擎中的海量数据主要来自原始数据、索引数据和日志数据三个方面。

(1) 原始数据。这些原始数据是爬虫服务夜以继日地在互联网中不间断下载的互联网数据, 包括文档、图片、视频等信息。

(2) 索引数据。索引服务会对爬虫服务下载的庞大数据进行数据分析, 以及建立索引, 而索引一般情况下均会超过原始数据的大小。

(3) 日志数据。每天数以亿计的用户请求源源不断地产生,为达到较好的搜索效果,日志服务记录下了所有用户请求信息,而这些日志数据也在日益增大。

面对原始数据、索引数据及日志数据，使得海量数据的存储不得不依靠大规模分布式存储技术，一般的分布式存储逻辑结构如图 2-7 所示。

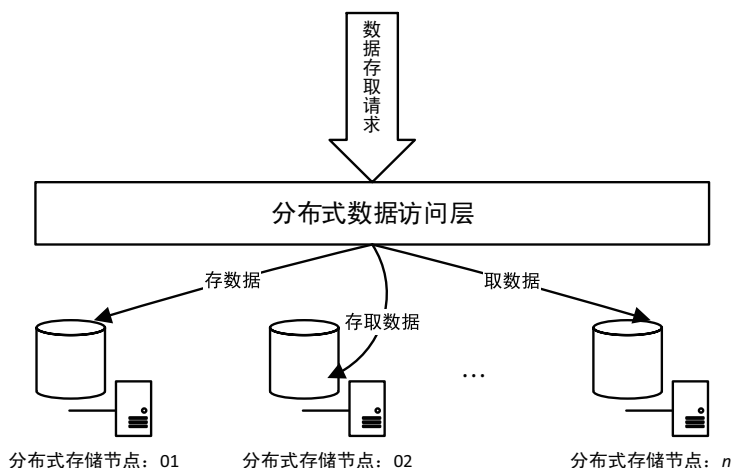


图 2-7 一般的分布式存储逻辑结构

海量数据的分布式存储中，不同数据可能会被存储到各个不同存储节点中。分布式存储也面临着非常重要的数据压缩、负载均衡、容灾还原等问题。

(1) 数据压缩。虽然庞大的数据被分布式存储到各个存储节点中，但是由于庞大的数据存储会导致企业硬件成本偏高，一般情况下各个节点存储数据时也会考虑对数据进行压缩。而压缩就要权衡压缩比和解压缩性能，倘若压缩比过高，则会导致解压缩性能变慢，对数据访问会造成影响。

(2) 负载均衡。在数据存储过程中，分布式集群中的每一个数据节点，都有可能产生数据倾斜现象，即少量数据存储节点中存储的数据量极大或者极小。需要采用各种方法来协调数据存储中的负载均衡问题。

(3) 容灾还原。任何存储节点都可能出现故障，一旦某部分数据出现故障，将导致该存储节点中的数据不可读。因此，整个分布式存储系统需要具备发生故障后，进行数据恢复的能力，并保证整个分布式存储系统稳定性一直处于可控范围内。

上述问题都是海量数据存储中必须解决的问题。

2.3.4 分布式计算

在海量数据面前，搜索引擎需要发掘里面的数据价值，仅仅通过普通的数据库分析很难达到理想效果，即使采用编程分析也是比较难的，因此需要一个计算平台协助搜索引擎进行数据分析。所以对于大数据的处理，需要引入分布式计算，分布式计算是采用多台服务器分散处理的方式，将由于数据过大而导致计算时间过长的任务拆解为更多的子任务，将这些子任务分别交给更多的服务器去处理，各个服务器处理完毕之后，将最终结果合并。整个过程中，从处理方式上可以分为流计算和批量计算，批量计算是先存储后计算，而流计算是直接计算。

从搜索引擎角度看，对这些海量数据的分析，不得不依赖分布式计算平台。但是部分信息采用的是离线批处理运算；对于部分信息的处理实时性越高越好，例如，针对数据抓取是离线运算，而对于结果排序需要采用实时计算。分布式计算之所以成为大数据处理分析的重要工具在于如下两个方面：

（1）高性能服务器也能进行复杂运算，但是高性能服务器不仅价格昂贵，而且一旦出现故障将导致众多问题。相反，将廉价的普通计算机组成一个分布式计算网络，不仅可以进行复杂运算，还可以避免单机故障导致系统瘫痪，具备高容错性。

（2）采用分布式计算可以有效将资源进行汇总，可以灵活扩展计算性能，维护成本低，并且可以让不同的分布式应用运行在分布式计算集群中，意味着任何开发者都可以向分布式计算平台发起执行应用的可能。

利用分布式计算，使得搜索引擎能够在尽可能短的时间内处理海量数据，分析出数据的价值，例如，在极短的时间内分析出当前实事热点新闻、为用户推荐最相关的广告、最短时间内为用户反馈搜索结果等。分布式计算和分布式



存储作为搜索引擎的基础平台，是后续一切工作的基础。

2.3.5 搜索排序技术

在搜索服务中，搜索服务接口通过搜索服务后台获得搜索结果，而在搜索服务后台向搜索服务接口返回结果之前，搜索服务后台有一项非常重要的工作即对结果排序，通俗意义上是对搜索结果中的文档集合进行排序。文档集合排序是搜索引擎中核心的组成部分之一，是用户直观感觉搜索引擎质量优劣的主要标准，时至今日各互联网公司依然在持续深入研究搜索引擎排序策略。

目前，主流的搜索引擎主要从利用搜索词、文档质量及用户信息三方面对搜索结果的文档集合进行排序。

(1) 利用搜索词进行排序。该过程又称为相关性排序，是根据用户的搜索词与文档本身的相关程度进行排序，整个过程只考虑文档与用户搜索词的匹配程度。

(2) 利用文档质量进行排序。文档的质量是一个多方面的评价标准，它包括对文档链接权值、文档链接质量的分析，对文档内容真实性、文档内容权威性等各方面情况计算一个合理参考值，给予文档基于质量上的评分。

(3) 利用用户信息进行排序。该过程又称为个性化排序，是根据用户搜索的历史记录、个性喜好、身份特征等方面为用户定制的搜索结果，即可能导致不同用户使用相同搜索词看到的搜索结果可能不一致。

对于用户搜索文档的最终排序结果是上述三类排序的综合排序，例如，三个文档：文档 A、文档 B、文档 C。在上述三类排序中均存在不同的排序序列，通过将三方面综合，给予一个最终排序序列，如图 2-8 所示。

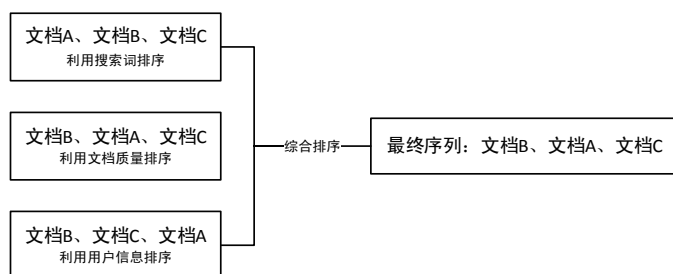


图 2-8 搜索文档最终排序与三类排序的综合关系

换个角度看搜索排序，除对搜索排序结果质量有很高的要求之外，对排序完成的时间也具有极高的要求，在极短的时间内完成排序也是搜索体验的重要影响因素之一。在上述三类排序中，利用搜索词和用户信息的排序方式采用分布式实时计算，而文档质量则在数据分析过程中已经完成，并不需要在用户搜索过程进行文档质量评分。

2.4 本章小结

本章从相对简易的角度去认识搜索引擎中的工作原理及大致使用到的技术体系。从基本模块结构上看，搜索引擎主要包括爬虫服务、索引服务、缓存服务、搜索服务及日志服务。这些服务相互之间产生关联影响。搜索引擎的所有服务均是由自然语言处理、知识图谱、海量数据存储、分布式计算、搜索排序等提供技术支持，这些技术支撑了搜索引擎的整个服务运作流程。读者通过对搜索引擎原理和技术的了解，有助于后续深入分析搜索引擎内部结构和实现。

第 3 章 自然语言处理框架

自然语言处理包括文本朗读、语音识别、分词、词性标注、句法分析、文本分类、自动摘要等技术范畴。在中文搜索领域，自然语言处理可以帮助搜索引擎深入认知中文语言表达的含义和意图。自然语言处理作为搜索引擎的技术之本、核心技术之一，几乎搜索引擎的每个技术环节都需要自然语言处理技术，也没有其他领域比搜索引擎更依赖于自然语言处理技术。

3.1 英文分词

分词是自然语言处理过程中对文本处理的最基本工作，是自然语言理解的基础。从语言学的角度讲，分词技术是将语句拆分成语句的各个组成的单元；从搜索引擎角度讲，分词技术是将长文本拆分为可理解的短文本信息，目的是更好地进行文本分析。在中文搜索引擎领域，分词主要包含英文分词和中文分词。

英文分词作为中文搜索领域中非常重要的一部分，它的分词效果也会对结果产生非常重要的影响。然而英文分词相对中文分词算法比较简单，首先只需要通过指定分隔符进行英文单词切分；其次是对切分好的单词进行单词还原（lemmatization），例如，将“got”还原为“get”；最后，可以选择是否移除停用词即可。

(1) 单词还原。单词还原涉及两个问题，一个是词形还原：把一个单词还原为单词的一般形式；另外一个词干提取，并不是指利用词形还原词典对英文单词进行词形还原。通过波特词干算法（Porter Stemmer）进行词干提取，词干提取是还原词语的修饰形式，得到单词最一般的写法形式，并不完全是词形还原，而是将词转换为词根，例如“fishing”“fished”“fish”和“fisher”为同一个词根“fish”通过转变而来，因此只需要将前四个单词还原为“fish”即可。

波特词干算法可以利用词库实现，也可以利用规则实现。在工程应用中，一般采用基于规则实现，首先处理单词复数形式及“ed”和“ing”结束的单词，例如，将“meetings”转换为“meet”；其次是如果单词中包含元音并且以“y”结尾，将“y”改为“i”；然后将双后缀的单词映射为单后缀；最后再处理类似“-ic-”“-full”“-ness”“-ant”“-ence”等后缀；波特词干算法中提取到的词干并不一定正确，但可以保证用户的输入和后台的处理是一致的。

(2) 停用词处理。在搜索引擎的搜索及数据分析过程中，为节省存储空间和提高效率，对被处理的自然语言数据（或文本）执行分析之前，会自动过滤掉一些字符或单词，这些词又称为停用词。原因在于停用词对于搜索引擎无法产生具体价值，对搜索结果也无法做到区分，与句子中的关键词恰好相反。

停用词并非自动生成，而是手动创建。这些停用词包括“is”“on”“at”等词汇，主要包括广泛无意词和无明确限定词。广泛无意词是表示被广泛使用的词汇，但是在实际搜索中不会产生实际意义的词。从词性角度讲，停用词是一些无明确限定词，涵盖语气助词、副词、介词、连接词等词语，这些词语对于句子的重要性需要在具体的语义环境中才能充分体现。过滤掉停用词，有助于搜索引擎减小搜索范围，减少搜索索引量和文档量，对于提升搜索整体性能，有着积极效果和作用。

对于英文分词，除按照空格正常分词的方法之外，还可以采用 N-Gram 分

词方法。N-Gram 是一种基于前后词语关系的语言模型，该模型表示当前词语仅与前面第 $N-1$ 个词语相关，而与其他词汇均不相关。采用 N-Gram 的分词方法，可以适当保持上下文关系。在 N-Gram 中，常常包括 Uni-Gram（一元组）、Bi-Gram（二元组）和 Tri-Gram（三元组），分别表示拆分句子为一个单词为单位、两个单词为单位、三个单词为单位。例如，对“welcome to the national conference center”进行 Uni-Gram、Bi-Gram、Tri-Gram 分词。移除停用词和词形还原之后句子为“welcome national conference center”，分词结果如表 3-1 所示。

表 3-1 “welcome national conference center” 的 Uni-Gram、Bi-Gram、Tri-Gram 分词结果

N-Gram	分词结果
Uni-Gram	“welcome” “national” “conference” “center”
Bi-Gram	“welcome national” “national conference” “conference center”
Tri-Gram	“welcome national conference” “national conference center”

值得说明的是，N-Gram 虽然主要应用于英文，但是也可以用于其他语言文字，尤其是对未知语言的文字进行分词，可以采用 N-Gram 进行默认分词。

3.2 中文分词

不言而喻，中文中最小的单位是字，但具有语义的最小单位是词，这是区分中文分词技术和英文分词技术中最重要的理论基础，也是导致切分方式不同的原因。中文分词技术是一项关键的技术，尽管目前的中文分词技术已经比较成熟，但是它依然在语义分析方向上有很大的发展空间，需要尽可能以自然语言的理解方式去分词。

3.2.1 中文分词概述

中文分词过程中不仅要做到语义分析，还包括多重性分词。多重性分词是

指分词结果尽可能多种多样。例如，句子“中国科学技术大学在哪？”这样一个简单的语句却包含了搜索中非常难处理的问题，正常情况下分词结果为：“中国科学技术大学\在哪\？”看似很完美的分词效果，但是对于大数据时代的搜索引擎来说，并不是理想的分词效果，为了提供更好的搜索结果，会做到如下分词：“中国科学技术大学\在哪\中国\科学\技术\大学\科学技术\”，这是分词的原子化，不仅描述全局，更深入到细节，这为搜索结果提供了很好的技术支持。另一方面，为了得到很好的结果，最好在进行语义分析的同时，分析用户搜索的意图，判断出原本用户最期待的答案是“安徽合肥”，但是这对自然语言处理的要求相对较高。

中文分词的方式有很多种，最常用的是基于词库的分词方式。但是对于搜索引擎使用的自然语言处理框架，仅仅采用基于词库的分词还是不够的，还需要利用机器学习的方式，采用基于上下文信息的分词技术，目前公认的机器学习方式能够达到的较好效果是基于条件随机场模型（Conditional Random Fields）的中文分词技术。

3.2.2 基于词库的分词技术

在词典的分词方法中，逆向最大匹配分词是常用的方式之一，在一般情况下，匹配效果较为满意，准确程度也在一定程度上依赖于词库。同理也存在正向最大匹配分词，这里的逆向最大和正向最大表示分词时允许每一次读取的最大文本长度。

正向最大匹配实质是在每次从头开始读取最大文本长度之后，去词库中查询该词是否存在，如果不存在，则减少一个字，再去词库中查询该词是否存在，依次类推，直到找到一个词为止。例如，假定某词库中包含“香港大学”“香港”“大学”“校庆”“典礼”等词。对于句子“香港大学校庆典礼”的正向

最大匹配顺序如图 3-1 所示。

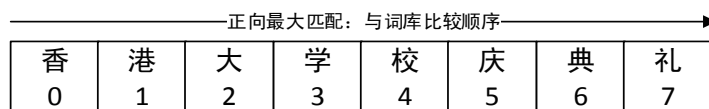


图 3-1 对于句子“香港大学校庆典礼”的正向最大匹配顺序

设定最大匹配长度为 5，则首先从字符串的开始位置依次取 5 个长度的词语“香港大学校”，将其与词库比较，若词库不存在“香港大学校”，则将“香港大学”与词库比较，结果命中词库，则意味着可以将“香港大学”视为一个词。依次将剩余字符串按照相同方法进行词库比较，最终分词结果为“香港大学 / 校庆 / 典礼”。

逆向最大匹配计算过程同正向最大匹配一致，唯一不同的是对匹配顺序的改变，匹配顺序如图 3-2 所示。

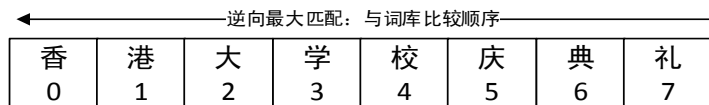


图 3-2 “香港大学校庆典礼”逆向最大匹配顺序

同样设定最大匹配长度为 5 的情况下，则从字符串的尾部位置向前依次取 5 个长度的词语“学校庆典礼”与词库进行比较，若不能匹配上词库则减少一个字符，将“校庆典礼”与词库进行比较，依次类推。最终分词结果与采用正向最大匹配分词方法的结果一致。

逆向最大匹配与正向最大匹配都各自拥有自己的优势，但是基于相同词典的情况下，工程应用表明，逆向最大匹配分词方式优于正向最大匹配分词方式。例如，针对句子“发展中国家领导人正在开会”设定正向最大匹配长度为 5，则按照正向最大匹配思想分词结果为：“/ 发展 / 中国 / 家 / 领导人 / 正在 / 开会”。然而按照逆向最大匹配分词结果为：“/ 发展 / 中 / 国家 / 领导人 / 正在 / 开会”。

当然并不是逆向最大分词方法时时刻刻都是最优的，逆向最大分词方法也存在效果差于正向最大匹配的情况，例如，“庞大数据”按照逆向最大分词方式会被分词为“庞 / 大数据”，而实际上“庞大 / 数据”略好。

在工程应用中，会将正向最大匹配分词方式与逆向最大匹配分词方式结合，俗称“双向匹配分词”。对于被分词的句子采用两种方法分别进行分词处理，然后将两者的分词结果进行比较，如果分词结果两者一致，则直接输出即可。如果不一致则按照如下原则优先输出。

(1) 分词结果中词越少越优先输出。例如，“三角形和平行四边形”，正向最大匹配分为“三角形 / 和平 / 行 / 四边形”，而逆向最大匹配分词结果为“三角形 / 和 / 平行四边形”，分别产生四个词语与三个词语，因此优先输出“三角形 / 和 / 平行四边形”。

(2) 分词结果在词库中能够找到的越多越优先显示。例如，“售后和服务”，按照最大正向匹配为“售后 / 和服 / 务”，词典中能够找到“售后”“和服”，而“务”不存在单独的词；按照逆向最大匹配为“售后 / 和 / 服务”，在词典中三者均能够找到。因此，优先选用逆向最大匹配结果“售后 / 和 / 服务”。

正向最大匹配与逆向最大匹配在无法通过上述两项原则区分优先输出结果时，则优先输出逆向最大匹配分词结果。

3.2.3 基于条件随机场的中文分词

采用条件随机场进行中文分词是一种有效的方式。条件随机场模型是基于最大熵马尔科夫模型（Maximum Entropy Markov Model, MEMM）及隐马尔科夫模型（Hidden Markov Model, HMM）为基础提出的一种判别式概率无向图学习模型，是一种用户标注和切分有序数据的条件概率模型，在使用条件随机场进行中文分词之前需要对判别式、概率无向图模型有一定了解。

(1) 判别式模型。判别式模型 (Discriminative Model) 是通过判别函数产生预测模型进行预测, 与之对应的是产生式模型。而产生式模型 (Generative Model) 是通过概率密度模型形成产生式模型进行预测, 如图 3-3 所示。

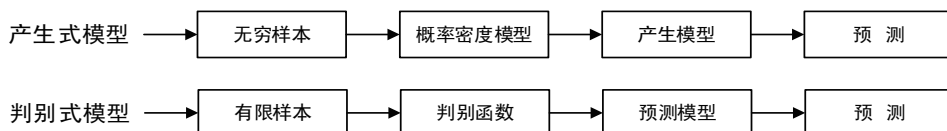


图 3-3 产生式模型与判别式模型的预测方式示例

对于输入数据 x , 判别类型标签 y , 产生式模型估计它们的联合概率分布 $P(x,y)$, 判别式模型估计条件概率分布 $P(y|x)$, 其中, 联合概率分布与条件概率分布都属于现代概率论中的概念, 两者的含义如表 3-2 所示。

表 3-2 联合概率分布与条件概率分布的含义

概率分布类型	含 义
联合概率分布 $P(x,y)$	从统计的角度去分析数据的分布情况, 对两个随机变量 X 和 Y , 其联合分布是同时对于 X 和 Y 的概率分布
条件概率分布 $P(y x)$	已知两个相关的随机变量 X 和 Y , 随机变量 Y 在条件 $\{X=x\}$ 下的条件概率分布是指当已知 X 的取值为某个特定值 x 时, Y 的概率分布

判别式模型和产生式模型是机器学习中的重要概念, 是条件随机场进行中文分词的理论基础。

通过示例说明判别式模型与产生式模型, 已知数据 (x,y) 表示 x 与 y 的某种特定关系, 对于多个数据 (b,a) 、 (b,a) 、 (c,a) 、 (c,b) , 分别通过产生式模型与判别式模型对 x 值和 y 值的关系进行分析。

通过产生式模型计算 $P(x,y)$, 则 $P(b,a)=1/2$ 、 $P(b,b)=0$ 、 $P(c,a)=1/4$ 、 $P(c,b)=1/4$ 。通过判别式模型计算 $P(y|x)$, 则 $P(a|b)=1$ 、 $P(b|b)=0$ 、 $P(a|c)=1/2$ 、 $P(b|c)=1/2$ 。

产生式模型与判别式模型的优缺点如表 3-3 所示。

表 3-3 产生式模型与判别式模型优缺点

模 型	优 点	缺 点
产生式模型	数据信息相对丰富，研究单类别问题灵活性较强。 能够充分利用先验数据。 模型可以通过增量学习的方式获得	学习过程相对比较复杂。 在目标分类的问题中容易产生较大的错误率
判别式模型	分类边界比较灵活，适用于多类别问题研究。 能够较好地分辨出类别之间的差异特征	不能反映训练样本的本身特性。 描述信息具备一定局限性

产生式模型可以依据贝叶斯公式转换为判别式模型，但是判别式模型不能转换为产生式模型。使用条件随机场模型进行分词，是因为它既具有判别式模型的优势，又具有产生式模型中转移概率，以序列化形式进行全局参数优化和解码的特点，解决了其他判别式模型中难以避免的标记偏见问题。在中文分词领域，条件随机场的效果优于隐马尔科夫模型和最大熵马尔科夫模型。

(2) 概率无向图模型。概率图模型是一种利用图的结构来表达随机变量之间条件独立性的概率模型，利用图中的每个节点表示随机变量，节点之间的边则用于表示随机变量之间的依赖关系。图中的边可以是有向的或者无向的，有向的则称为概率有向图模型，无向的则称为概率无向图模型。概率有向图模型如图 3-4 所示，通过图的方式将相互之间的影响关系表示出来，天气状况和环境温度会影响到感冒，感冒是导致头晕的一个因素，边上的权值为影响的概率值，也可以理解为两个节点之间的联合概率分布。

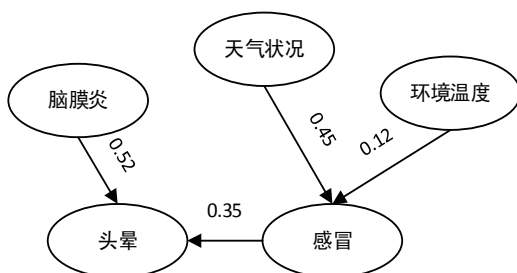


图 3-4 概率有向图模型



无向图模型又称为马尔科夫随机场，是一组具备马尔科夫性质、随机变量的全联合概率分布模型。这里的马尔科夫性质是指当一个随机过程在给定现在状态及所有过去状态的情况下，其未来状态的条件概率分布仅依赖于当前状态。

(3) 条件随机场模型。在理解判别式模型及概率图模型之后，则可以较好地理解条件随机场的概念。条件随机场模型对应一个无向图 $G=(V,E)$ ， $Y=(Y_v)$ $v \in V$ ， Y 中的元素与无向图的顶点一一对应。当在条件 X 下，随机变量 Y_v 的条件概率分布符合图的马尔科夫性质，即可称当前 (X, Y) 是一个条件随机场。

条件随机场模型是通过给定需要标记的观察序列的条件下，计算整个标记序列的联合概率分布。从中文分词角度分析，每个词语中的每个字都存在四种可能的状态，分别是词头(Begin)、词中(Middle)、词尾(End)和单字成词(Single)，简称 B、M、E、S。给定句子中的每个字则被视为条件随机场模型中的观察序列，B、M、E、S 是需要对每个字进行标记的状态，整个句子则会形成一个标记序列，条件随机场求解的即是句子中字的标记序列的联合概率分布。

采用条件随机场模型进行分词，需要通过如下步骤完成：

(1) 准备语料库。准备一个已经分好词的语料库，用于条件随机场模型的参数学习，如下所示，语料库中由大量类似下面句子组成，词语之间采用空格符分隔。

尽管印尼中央和地方政府已派出上千人的灭火队，但由于该地区长期干旱少雨，所以火势至今未得到有效控制。

(2) 初步语料库特征学习。需要在语料库中的每个词组中分析出每个字的状态，例如“收益”需要改进为“收|B 益|E”。通过将语料库的每个分好的词，添加其状态信息，标记后的语句如下所示。

尽|B 管|E 印|B 尼|E 中|B 央|E 和|S 地|B 方|M 政|M 府|E 已|S 派|B 出|E 上|B 千|E 人|S 的|S 灭|B 火|E 队|S，|S 但|S 由|B 于|E 该|S 地|B 区|E 长

|B 期 |E 千 |B 旱 |E 少 |S 雨 |S , |S 所 |B 以 |E 火 |B 势 |E 至 |B 今 |E 未 |S 得 |B 到
|E 有 |B 效 |E 控 |B 制 |E 。 |S

使用这样的语料库，即可将中文分词问题转换为序列标注问题，对一个未分词的句子中每个字进行类似上述的序列标注（标注状态 B、M、E、S）即可知道分词结果。例如，对“我们爱中国”进行状态标注后结果为“我 |B 们 |E 爱 |S 中 |B 国 |E”，表示将句子“我们爱中国”分词为“我们 爱 中国”。

（3）词语特征学习。词语特征学习是整个过程中非常重要的部分，词语的特征包括如下信息：

第一，针对特定的某个字，它共在语料库中出现的次数值。例如，“我”字共在语料库中出现了 256 次。

第二，针对具体的某个字，计算它的状态为词头（B）、词中（M）、词尾（E）、单字成词（S）的概率值。例如，在 256 次出现“我”字的时候，“我”作为词头的概率值为 0.8，在词中的概率值为 0.01，依次计算所有出现的字。

第三，针对某个字，计算当它的状态为词头（B）时，它转移到下一个词的状态概率值。每个字都有属于自己的状态，但是这个字的后面一个字也有属于自己的状态，那么需要计算每一个字的状态到下一个字的状态（或许是 B、M、E、S 之一）的概率值。例如，“我”字，当“我”的状态为 B 的时候，后面跟的字中，状态为 B 的为 0 个，状态为 M 的为 10 个，状态为 E 的为 20 个，状态为 S 的为 0 个。依次统计“我”的状态为 M、E、S 的时候，下一字的状态，此过程计算的是状态之间相互的转移概率。针对四个状态，因此会形成一个 4×4 的矩阵，矩阵中的值是它们相互之间的转移概率值。

截至前面三个特征学习，似乎条件随机场与隐马尔科夫模型的方式并不存在太大差异，但是从理论上研究，基于条件随机场的分词准确率一定高于基于隐

马尔科夫模型的分词准确率。原因在于条件随机场会利用文本的上下文关系作为分词的一个参考信息。相比隐马尔科夫模型，条件随机场多计算了当一个字出现的时候，记录它前一个字及后一个词出现的内容，并以概率的方式记录。

第四，针对某个字出现的时候，需要计算该字的下一个字出现的内容，并计算出两字同时出现的概率值。例如，状态为 B 的“我”，下一个字是“们”的概率为 67.9%；状态为 S 的“我”，上一个字是“的”的概率为 21%；状态为 B 的“我”下一个字是“爱”的概率为 17.8% 等。记录每一个字在四种状态下的上下文关系，是非常重要的步骤。这一步中，仅仅记录了上一个字和下一个字的上下文关系，条件允许的情况下，可以记录上两个字和下两个字的上下文关系或者更多上下文关系。

(4) 开始分词。既然特征已经训练好了，则可以通过已经训练好的参数值实现中文分词。例如，对用户输入的句子“希腊的经济结构较特殊”进行中文分词处理。

第一步，将“希腊的经济结构较特殊”变成字符数组。即“希”“腊”“的”“经”“济”“结”“构”“较”“特”“殊”。

第二步，取出每个字对应特征，并根据状态信息，绘制字与状态的初始矩阵映射关系表，如表 3-4 所示，在分词前的初始状态概率值为 0。

表 3-4 字与状态的初始矩阵映射关系表

字	状态 (B)	状态 (M)	状态 (E)	状态 (S)
希	0	0	0	0
腊	0	0	0	0
的	0	0	0	0
经	0	0	0	0
济	0	0	0	0
结	0	0	0	0
构	0	0	0	0

续表

字	状态 (B)	状态 (M)	状态 (E)	状态 (S)
较	0	0	0	0
特	0	0	0	0
殊	0	0	0	0

根据表 3-4，既然是矩阵并且是求矩阵里面的一个路径，那么很容易想到维特比算法，维特比算法是一种动态规划算法。因此，只需计算出“希”字在 B、M、E、S 的状态值，后面的问题即可游刃有余。

设定矩阵中的值为 S ， $S[\text{字}][\text{当前状态}] = \text{MAX}(P[\text{上一个字的状态}][\text{当前状态}] * S[\text{上一个字}][\text{任何一个状态}] + W[\text{前（后）一个字_当前状态}][\text{当前的字}] + R[\text{当前状态概率}])$ （备注： R 是特征二， P 是特征三， W 前是特征四的上文部分， W 后是特征四的下文部分。）

例如，针对表 3-4 中的“腊”字，在为状态 B 上的值为： $S[\text{腊}][B] = \text{MAX}(P[B][B] * S[\text{希}][B], P[M][B] * S[\text{希}][M], P[E][B] * S[\text{希}][E], P[S][B] * S[\text{希}][S]) + W[\text{前}][\text{希_B}][\text{腊}] + W[\text{后}][\text{的_B}][\text{腊}] + R[B]$ ，同理可以计算 $S[\text{腊}][M]$ 、 $S[\text{腊}][E]$ 、 $S[\text{腊}][S]$ 。依次类推，可以计算出后面其他字的情况。由于“希”字出现在首字，不存在其他状态转移到其状态的概率，因此， $S[\text{希}][B] = W[\text{前}][_B][\text{希}] + W[\text{后}][\text{腊_B}][\text{希}] + R[B]$ ，依次计算出“希”字的其他值。上述过程，通过计算后的矩阵值如表 3-5 所示。

表 3-5 字与状态对于关系计算后的矩阵值

字	状态 (B)	状态 (M)	状态 (E)	状态 (S)
希	1.828 402 366 863 905 1(0)	0.082 840 236 686 390 54(0)	0.013 313 609 467 455 622(0)	0.010 355 029 585 798 817(0)
腊	0.059 582 893 589 493 494 (2)	0.408 474 093 043 961 04(0)	2.830 977 681 853 876 3(0)	0.083 451 501 464 701 27(2)
的	0.944 853 777 054 963 5(2)	0.034 887 252 751 967 93 (1)	0.039 550 037 001 873 96(1)	2.008 219 658 674 28(2)
经	2.849 596 792 789 322 (3)	0.227 192 059 628 115 42 (0)	0.113 034 639 833 597 86(0)	0.318 765 430 922 945 57(3)

续表

字	状态 (B)	状态 (M)	状态 (E)	状态 (S)
济	0.032 237 979 582 153 37(2)	0.461 670 498 344 681 55(0)	3.528 501 575 733 457 (0)	0.005 930 136 782 490 121 (2)
结	3.021 672 740 287 898 2(2)	0.073 017 008 094 566 13(1)	0.192 961 888 587 480 15(1)	0.918 542 489 234 858 4(2)
构	0.171 431 949 444 147 25(2)	0.070 519 255 836 643 65(0)	3.410 449 094 696 205 4(0)	0.023 518 184 506 250 442 (2)
较	1.585 547 884 869 041 9(2)	0.002 419 778 386 551 498 (1)	0.314 324 003 725 793 44(0)	1.865 184 802 597 638 3(2)
特	0.777 854 543 075 236 4(3)	0.225 740 377 774 251 4(0)	1.015 124 300 735 784 4(0)	0.211 061 978 623 756 59(3)
殊	0.154 136 218 162 806 2(2)	0.173 723 789 513 471 45(0)	1.186 742 536 054 932 2(0)	0.106 532 644 697 739 29(2)

在表 3-5 所示矩阵中所有值的后面，都有一个小括号，里面存放的是路径，也就是这个值是通过上一个字的其中某一个状态值过来的（因为有计算 max 的过程，就会有路径选择），记录下来之后，以便于路径回溯。

当得到这个矩阵之后，分词系统只需将“殊”中的最大值取出，即为 1.186 742 536 054 932 2 对应状态是 E (End)，来自上一字的状态 0，也就是“特”字的状态 B (Begin)，再次查看“特”字来自上一个字“较”的状态 S (Single)。依次进行路径回溯，得到标注后的状态为“希|B 腊|E 的|S 经|B 济|E 结|B 构|E 较|S 特|B 殊|E”，转换成分词结果就是“希腊 的 经济 结构 较 特殊”，完成分词过程。

虽然基于条件随机场模型已经可以完成中文分词，但是往往也会利用词典辅助分词，原因在于采用词库分词的方式可以灵活地对分词结果进行改变，而基于条件随机场的分词方式，若要手动改变一个分词结果相对较困难。

3.2.4 分词粒度

一个分词器没有绝对的优劣，不同的评价标准也会导致评价结果不一致。

并且分词本身也是具有歧义的，由于分词粒度不一致，大家对分词结果的评价也会不一致。例如，“国家科学技术委员会”，粗粒度则分为“国家科学技术委员会”，细粒度则分为“国家科学技术委员会”，两种不同分词粒度的结果均可视为正确的分词结果。但若分词中太多词汇因为粒度不一致导致不同的分词结果，则会有一定影响。

分词粒度对搜索引擎的影响是：分词粒度越大，则分词后的词数量越多，间接也会导致索引存储量越大，搜索展示结果越多，进一步导致搜索结果的相关性越差。反之，分词粒度越小，最终得到的词数量则越少，也使得索引存储量越小，但间接也会导致搜索展示结果偏少，但搜索结果的相关性较好。

除减少分词粒度导致的分词结果不一致之外，还需要减少分词的歧义。例如，对句子“湖南省长沙河镇长”分词结果可能为“湖南省长沙河镇长”或者“湖南省长沙河镇长”，这样的分词结果很可能导致搜索结果相关性较差，类似的还有“张三打死狗”“重庆市长江大桥”“羽毛球拍卖完了”等。

3.3 词性标注

词性标注是自然语言中同分词技术一样重要的技术。它是给分词结果中所有的词标注其词性特征，这些词性包括名词、动词、副词等。对中文领域进行词性标注相对比较简单，由于中文语言特征，大多数词语都只有一个词性，尤其是专业词汇，笔者曾经实验，通过给分词结果中的所有词赋予词语最常用的词性，也能达到 75% 的准确率，但是为了达到更加精准的词性标注效果，可以采用基于统计学习的方法进行词性标注，而这种常用的方法一般可以采用隐马尔科夫模型对词语进行词性标注。

3.3.1 隐马尔科夫模型概要

隐马尔科夫模型是一种概率有向图模式，它主要包含隐藏状态、可观察对象、初始状态概率矩阵、隐藏状态转移概率矩阵等五个描述参数。

- (1) 隐藏状态 S 。表示无法通过直接观察得到的状态信息。
- (2) 可观察对象 O 。表示通过直接观察可以得到的序列信息。
- (3) 初始状态概率矩阵 π 。表示隐藏状态在最开始及结束时的状态概率。
- (4) 隐藏状态转移概率矩阵 A 。隐藏状态相互之间的转移概率。

(5) 观测状态转移概率矩阵 B 。可观察的序列都拥有对应的隐藏状态，这种对应关系可以通过概率的方式体现。

一般可以采用 $\lambda=(A,B,\pi)$ 三元组表示隐马尔科夫模型，因此，隐马尔科夫模型实质上是标准马尔科夫模型的扩展形式，不同之处在于新增了可观察序列集合和可观察序列与隐藏状态之间的概率集合。如图 3-5 所示， S_1 、 S_2 、 S_3 分别表示隐藏状态， O_1 、 O_2 、 O_3 表示可观察对象序列、 A_{12} 、 A_{23} 表示隐藏状态之间的转移概率， B_1 、 B_2 、 B_3 表示可观察对象序列到隐藏状态的转移概率。

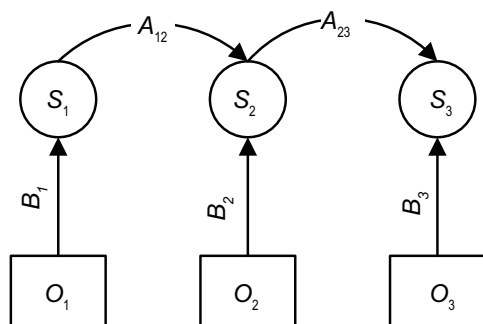


图 3-5 隐马尔科夫模型的概念图

隐马尔可夫模型是一种统计模型，它被用于分析一个含有隐含未知参数的

马尔科夫过程。而马尔科夫过程是一类随机过程，表示在目前已知状态条件下，它未来的改变不依赖于它以往的改变。隐马尔科夫模型在解决问题时，主要分为评估、解码和学习。

(1) 评估。给定一个隐马尔科夫模型，求其一个观察序列的概率，已知模型参数 $\lambda=(A,B,\pi)$ 求解某一观察序列 $O=o_1o_2o_3\cdots o_n$ 的概率，一般采用前向算法解决。

(2) 解码。给定一个观察序列，求出其最大可能的隐藏状态序列，已知模型参数 $\lambda=(A,B,\pi)$ 求解某一观察序列 $O=o_1o_2o_3\cdots o_n$ 的隐藏状态 S 的序列。一般采用 Viterbi 算法解决。

(3) 学习。根据观察序列生成隐马尔科夫模型，一直观察序列 $O=o_1o_2o_3\cdots o_n$ ，寻找隐马尔科夫模型参数 $\lambda=(A,B,\pi)$ ，使得观察序列 O 的概率最大，一般采用向前向后算法。

3.3.2 隐马尔科夫模型与词性标注

中文词性标注属于解码问题，因此只需对隐马尔科夫模型进行参数训练，然后通过维特比算法计算出最可能的隐藏状态序列即可。

词性标注的语料库如下所示，隐马尔科夫模型的 5 个参数均可从下面示例中提取出。

2005 年 5 月 1 日 |t 谢亚冰 |nr 担任 |v 陪审员 |nnt 9 年 |t 期间 |f 为了 |p 充实
|v 自己 |rr 法律 |n 知识 |n 谢亚冰 |nr 完成 |v 中央电大 |nis 法律 |n 专业 |n 自学 |vn

将中文词性标注与隐马尔科夫模型的参数结合，因此隐马尔科夫模型 5 个参数在中文词性标注中的含义如表 3-6 所示。

表 3-6 隐马尔科夫模型 5 个参数在中文词性标注中的含义

隐马尔科夫模型参数	词性标注中含义	示 例
隐藏状态 S	词的状态词性集合	nr、p、qt...
可观察状态 O	所有语料库中的词集合	路甬祥、于、1942 年...
初始状态概率矩阵 π	词中各种隐藏状态的初始概率	
隐藏状态转移概率矩阵 A	词性之间的相互转移概率	$P(nr,p),P(p,qt),P(qt,vi)\cdots$
观测状态转移概率矩阵 B	每一个词到各自词性的概率	$P(\text{路甬祥},nr),P(\text{于},p)\cdots$

上面 5 个参数在词性标注中的含义均可基于语料库统计分析得出。

在基于隐马尔科夫模型的词性分析中，隐藏状态 S 即词性状态集合。词性状态比中文分词中的状态包含更多。中文分词仅包含词头、词中、词尾和单字成词，而词性状态则包括名词、动词、形容词、副词等。

(1) 名词 (n)。名词是一个比较广泛的词性，可细分为如表 3-7 所示的更多状态。例如，句子“谢亚冰 |nr 担任 |v 陪审员 |nnt”中词语“谢亚冰”对应的词性状态即为名词。

表 3-7 名词细分状态

名词状态	简 写	名词状态	简 写
汉语姓氏	nr1	汉语名字	nr
日语人名	nrj	音译人名	nr _f
地名	ns	音译地名	ns _f
机构团体名	nt	其他专名	nz
名词性惯用语	nl	名词性语素	ng

(2) 时间词 (t)，时间词性语素 (tg)。

(3) 处所词 (s)，方位词 (f)。

(4) 动词 (v)。动词也是较大的词性类型，其中包括副动词、趋向动词、名动词等。动词细分状态如表 3-8 所示。

表 3-8 动词细分状态

动态状态	简 写	动词状态	简 写
副动词	vd	名动词	vn
动词“是”	vshi	动词“有”	vyou
趋向动词	vf	形式动词	vx
不及物动词（内动词）	vi	动词性惯用语	vl
动词性语素	vg		

（5）形容词（a）。包含副形容词（ad）、名形词（an）、形容词性语素（ag）和形容词性惯用语（al）。

（6）区别词（b），区别词性惯用语（bl）。

（7）状态词（z）。

（8）代词（r）。代词细分状态广泛，如表 3-9 所示。例如，句子“充实|v 自己|rr 法律|n 知识|n”中词语“自己”的词性状态为人称代词。

表 3-9 代词细分状态

代词状态	简 写	代词状态	简 写
人称代词	rr	指示代词	rz
时间指示代词	rzt	处所指示代词	rzs
谓词性指示代词	rzv	疑问代词	ry
时间疑问代词	ryt	处所疑问代词	rys
谓词性疑问代词	ryv	代词性语素	rg

（9）数词（m）、数量词（mq）。

（10）量词（q）、动量词（qv）、时量词（qt）、副词（d）。

（11）介词（p）、介词“把”（pba）、介词“被”（pbei）。

（12）连词（c）、并列连词（cc）。

（13）助词（u）。助词细分状态如表 3-10 所示。

表 3-10 助词细分状态

助词状态	简 写	助词状态	简 写
助词“着”	uzhe	助词“了”，“喽”	ule
过	uguo	的底	ude1
地	ude2	得	ude3
所	usuo	等等 云云	udeng
一样 一般 似的 般	uyy	的话	udh
来讲 来说 而言 说来	uls	之	uzhi
连（“连小学生都会”）	ulian		

(14) 叹词 (e)。

(15) 语气词 (y)。

(16) 拟声词 (o)。

(17) 前缀 (h)。

(18) 后缀 (k)。

(19) 字符串 (x)。

(20) 非语素词 (xx)。

(21) 网址 URL (xu)。

(22) 标点符号 (w)。标点符号细分状态如表 3-11 所示。

表 3-11 标点符号细分状态

标点符号状态	简 写	标点符号状态	简 写
左括号，全角：([{ 《 【 ‹ 半角： ([{ <	wkz	右括号，全角：)] } 》 】 › 半 角：)] { >	wky
左引号，全角：“ ‘ 『	wyz	右引号，全角：” ’ 』	wyy
句号，全角：。	wj	问号，全角：？ 半角：?	ww
叹号，全角：！ 半角：!	wt	逗号，全角：， 半角：,	wd
分号，全角：； 半角：;	wf	顿号，全角：、	wn
冒号，全角：： 半角：:	wm	省略号，全角：…… …	ws
破折号，全角：—— — — — — 半角：--- ----	wp	百分号千分号，全角：% ‰ 半 角：% ‰	wb

续表

标点符号状态	简 写	标点符号状态	简 写
单位符号，全角：¥ \$ £ ° ℃ 半角：\$	wh		

隐马尔科夫模型解决词性分析问题，实质是对每一个词隐藏的的词性求最大联合概率密度问题。例如，给定句子“对口资源优先合理安排”，通过分词得到结果“对口 资源 优先 合理 安排”，因此，隐马尔科夫模型则是对句子中的每一个词对应的词性求解最大联合概率密度，如图 3-6 所示，词语“对口”包含词性“vd”“vn”“v”，词语“资源”包含词性“n”，每一个词语的词性之间产生有向依赖关系形成一个概率有向图模型。

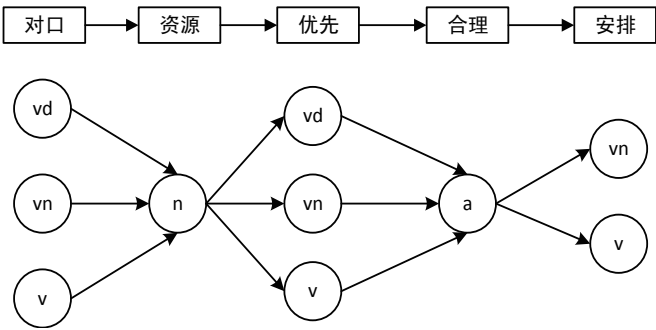


图 3-6 隐马尔科夫模型对词性分析问题的有向概率图模型

图 3-6 的概率有向图模式中的边，即为隐马尔科夫模型的隐藏状态转移概率，是整个隐藏状态转移概率矩阵中的一部分，表 3-12 所示为隐藏状态的转移概率矩阵。

表 3-12 隐藏状态的转移概率矩阵（局部）

	n	vd	vn	v	a
n	0.45	0.12	0.13	0.25	0.23
vd	0.06	0.10	0.20	0.16	0.12
vn	0.11	0.26	0.23	0.31	0.10
v	0.39	0.13	0.16	0.21	0.33
a	0.11	0.15	0.2	0.28	0.05



将有向概率图与隐藏状态转移概率矩阵结合后形成的转移概率图如图 3-7 所示。

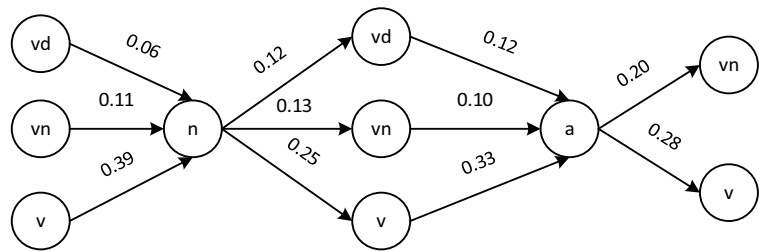


图 3-7 词性之间的转移概率图

除此之外，还需要观测对象的转移概率矩阵，表 3-13 所示为观察对象到词性的概率矩阵，对应图 3-7 所示为图中每一个节点出现的概率。

表 3-13 观察对象到词性的概率矩阵

	词性 (n)	词性 (vd)	词性 (vn)	词性 (v)	词性 (a)
对口	0	0.26	0.30	0.35	0
资源	0.75	0	0	0	0
优先	0	0.34	0.35	0.36	0
合理	0	0	0	0	0.8
安排	0	0	0.30	0.32	0

为了更好地表达图结构，引入虚拟节点“开始”和“结束”，作为概率有向图的首和尾，并将观察对象的状态转移矩阵与初始概率矩阵结合。虚拟节点“开始”与图的关系则与词性的初始概率矩阵有关，以“开始”作为图的出发点，指向词语“对口”拥有的词性。词性的初始概率矩阵如表 3-14 所示。

表 3-14 词性的初始概率矩阵

初始概率	n	vd	vn	v	a
开始	0.30	0.12	0.16	0.18	0.13
结束	0.35	0.21	0.14	0.25	0.09

结合词性的初始状态概率及观察对象的转移概率矩阵，图 3-8 所示为构建完整的隐马尔科夫模型的词性分析模型图。

表 3-16 利用维特比算法求解词语“对口”与词性概率值

	词性 (n)	词性 (vd)	词性 (vn)	词性 (v)	词性 (a)
对口	0	0.0312	0.048	0.063	0

但是不同点在于计算词语“资源”时,需要考虑上一个词语的词性到词语“资源”当前词性的转移概率。

例如,针对 $S[“资源”][“n”]$ 的计算如下所示:

$$\begin{aligned}
 S[“资源”][“n”] &= \max(P[“n”][“n”] * S[“对口”][“n”], P[“vd”][“n”] * \\
 &S[“对口”][“vd”], P[“vn”][“n”] * S[“对口”][“vn”], P[“v”][“n”] * \\
 &S[“对口”][“v”], P[“a”][“n”] * S[“对口”][“a”]) + P[“资源”][“n”] \\
 &= \max(0 * 0.45, 0.0312 * 0.12, 0.048 * 13, 0.063 * 0.25) + 0.75 \\
 &= 0.01575 + 0.75 \\
 &= 0.76575
 \end{aligned}$$

根据上述过程依次类推,最终完整的结果如表 3-17 所示,值得注意的是,需要将求 max 最大值的上一个词性记录下来。

表 3-17 利用维特比算法求解词语词性的最终概率矩阵

	词性 (n)	词性 (vd)	词性 (vn)	词性 (v)	词性 (a)
对口	0	0.0312	0.048	0.063	0
资源	0.76575(v)	0.01248(vn)	0.01104(vn)	0.01488(vn)	0.02079(v)
优先	0.3445875(n)	0.43189(n)	0.4495475(v)	0.5514375(n)	0.1761225(v)
合理	0.215060625(v)	0.11688235(vn)	0.103395925(vn)	0.139359725(vn)	0.981974375(v)
安排	0.10801718125(a)	0.14729615625(a)	0.496394875(a)	0.594952825(a)	0.04946394375(n)

根据表 3-17,首先找到最后一个词语对应的最大可能的词性;然后根据路径回溯,分别找出每一个词语的对应词性;最后将词语与词性状态综合。因此对于句子“对口资源优先统筹安排”词性标注结果为“对口 /v 资源 /n 优先 /v 合理 /a 安排 /v”。

同中文分词相比，词性状态相对较多，但是通过维特比算法，依然可以计算各个词语隐藏的词性状态，最终完成词性标注。

3.4 语义相似度

词语的语义相似度是研究特定环境下词语的相似性。在中文语义分析的应用中，需要用数值的方式来量化词语具体相似程度。目前常见的词语相似度计算方法包括根据世界知识与分类体系计算，或是通过语料库统计学习的方式获得。早在很久以前，有研究员就提出，两个词语的相似度取决于词语之间的共性（Commonality）和个性（Differences），并且给予一个参考公式：

$$\text{Sim} = (A, B) \frac{\log(\text{Common}(A, B))}{\log(\text{Diff}(A, B))}$$

根据上述公式对语义相似度的表达则意味着语义相似度与共性成正比、与个性差异成反比。然而需要去确定词语之间的共性和个性是一个比较难的问题，因此学术界提出了基于义原的概念，来表达词语的共性和个性。

《知网》（英文名称：HowNet）是一个以中文与英文的词语概念描述集，用于表示词语概念之间的深层次关系，这里的概念即指语义。一个词语可能存在多个语义。对于语义的表述通过义原进行描述，义原是语义最小的语义计算单位。义原之间的关系也有很多，例如，上下位关系、同义关系、反义关系等。《知网》目前大概有 1 500 个义原，这些义原大致分为：

- （1）基本义原。Event| 事件、Entity| 实体、Attribute| 属性值、aValue| 属性值、quantity| 数量、qValue| 数量值、SecondaryFeature| 次要特征。
- （2）语法义原。syntax| 语法。

(3) 关系义原。EventRole| 动态角色、EventFeatures| 动态属性。表 3-18 所示为一些词语的义原描述。

表 3-18 一些词语的义原描述

词 语	词 性	义原描述
东汉	N, 名词	time 时间 ,#royal 皇 ,ProperName 专 ,past 昔 ,(China 中国)
克里米亚	N, 名词	place 地方 ,ProperName 专 ,(Russia 俄罗斯)
墨守成规	ADJ	aValue 属性值 ,behavior 举止 ,stiff 呆 ,undesired 莠
东三省	N, 名词	place 地方 ,provincial 省 ,mass 众 ,ProperName 专 ,(China 中国)
动荡	ADJ	aValue 属性值 ,circumstances 境况 ,dangerous 危 ,undesired 莠
动荡	V	uprise 暴动 ,politics 政

表 3-18 中，每一条记录称为一个义项，一个词可能有多个义项，每个义项的描述由多个义原组成，以逗号分隔。义原是用来补充解释义项的，义原中可能存在一个特殊符号：“#”表示相关，“^”表示不存在相关的可能。

约定两个词语的相似度是各个义项相似度的最大值。例如，针对词语“克里米亚”和“东三省”的相似度计算，实质是求描述中各个义项的相似度，如下公式所示。

$$\text{Sim}(W_{\text{克里米亚}}, W_{\text{东三省}}) = \max(\text{Sim}(S_{\text{克里米亚}}, S_{\text{东三省}}))$$

上面将词语的相似度归结于义项的相似度，但是义项最终又归结于义原来表示，因此，义原的相似度计算是计算词语相似度的基础。而义原又是通过树状的义原语义层次树来表示的，如图 3-9 所示。

可以通过计算义原与义原之间在义原语义层次树中的路径距离表示语义相似度。距离越近，则相似度越高，以 P 表示义原，计算 P_1, P_2 义原相似度的公式如下所示，其中 d 表示 p_1, p_2 在义原语义树中的路径长度， a 是调节参数。

$$\text{Sim}(p_1, p_2) = \frac{a}{d+a}$$

《知网》与传统的其他语义词典不同的是，它通过义原将词语包含的概念给描述出来，其他词典则通过树状结构将语义映射到树的体系中，通过树的距离、共享根节点来表达语义相似。因此， $S_{\text{克里米亚}} = \{P_{\text{地方}}, P_{\text{专}}, P_{\text{俄罗斯}}\}$ ，而 $S_{\text{东三省}} = \{P_{\text{地方}}, P_{\text{省}}, P_{\text{众}}, P_{\text{专}}, P_{\text{中国}}\}$ ，最终通过计算得出词语“克里米亚”与“东三省”的语义相似度为 0.8。

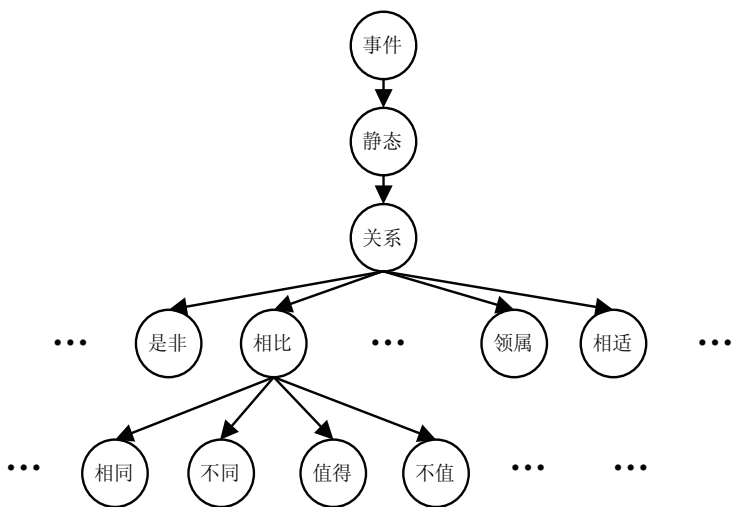


图 3-9 义原语义层次树

3.5 依存句法分析

依存句法分析在机器翻译中经常用到，主要是针对句子和短语进行结构化分析，用于确定句子中的词与词、短语与词等之间的相互关系，并利用树结构进行层次化表达，它是自然语言处理的关键问题之一。

3.5.1 依存句法分析概要

从搜索引擎角度，依存句法分析可以使搜索引擎更加准确地理解文本信息，

例如，确定句子中的词语相互关系及特定修饰性词语，尤其是智能搜索中，对句子的理解往往决定了数据理解的方向。例如，对于搜索“马云的淘宝网创办于什么时候”，通过句法分析，可以得到搜索句子的搜索重心是词语“创办”而不是“马云”或者“淘宝网”，“马云”“淘宝网”仅仅是一个核心成分的修饰词而已，用户期望的搜索结果答案也与“马云”没有直接关系，而“淘宝网创办”才是整个搜索的核心。

依存句法分析需要通过机器学习的方式进行分析。以清华大学汉语均衡语料库 TH-ACorpus 中的中文语义依存关系分析语料库为例，如表 3-19 所示。

表 3-19 依存句法分析语料库

词序号	词语	词语原型	粗粒度词性	细粒度词性	句法特征	当前词语的中心词	当前词语与中心词关系
1	伊斯兰	伊斯兰	n	nsf	—	2	限定
2	世界	世界	n	n	—	6	限定
3	的	的	u	ude1	—	2	“的”字依存
4	又	又	d	d	—	5	评论
5	一	一	m	m	—	6	数量
6	奇迹	奇迹	n	n	—	0	核心成分

表 3-19 所示为语料库内容的一般形式，也可以将其表达为依存句法关系图和依存句法关系树，分别如图 3-10 和图 3-11 所示。

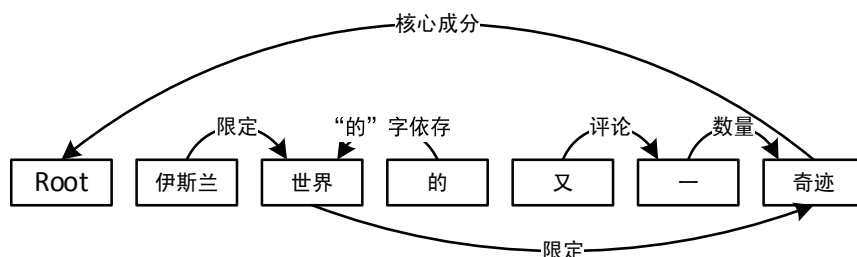


图 3-10 “伊斯兰世界的又一奇迹”依存句法关系图

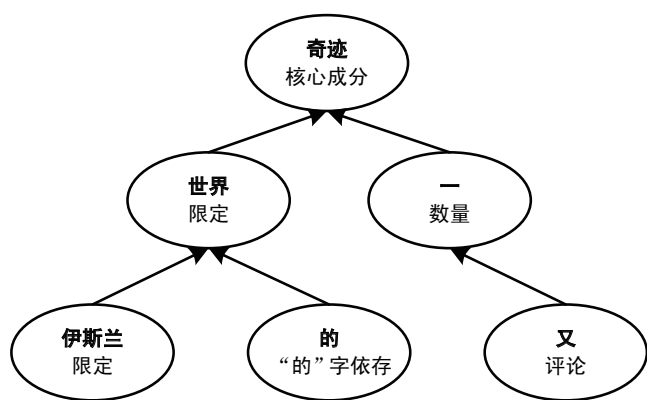


图 3-11 “伊斯兰世界的又一奇迹” 依存句法关系树

“马云的淘宝网创办于什么时候” 的依存句法关系树如图 3-12 所示。

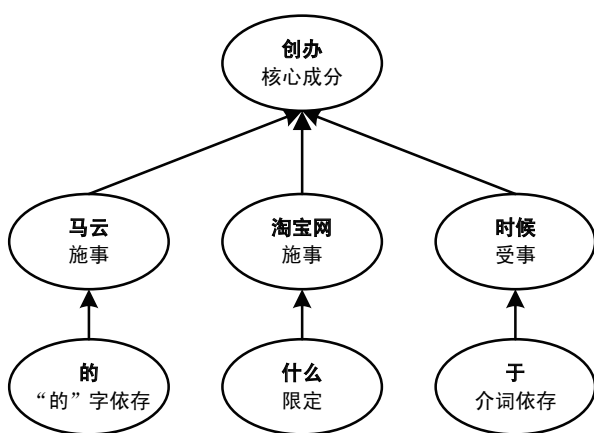


图 3-12 “马云的淘宝网创办于什么时候” 的依存句法关系树

语料中依存关系包括主语义关系、辅助语义关系、定状语语义关系、连动词及从句语义关系、特殊句法结构、特殊关系，如表 3-20 所示。

表 3-20 语料库中依存关系种类集合

主语义关系	施者	关系主体	经验者	描写体	存现体
	领有者	内容	类指	占有物	目标
	受事	整体	部分	代价	触及部件

续表

辅助语义关系	时间 后延时段 处所 终状态 相伴体 根据 比较内容	进程时间 原处所 参照体 来源 比较量	起始时间 通过处所 目的 工具	终止时间 终处所 原因 手段	时距 原状态 方向 材料
定状语语义关系	限定 评论 频率	数量 方式	描述 程度	同位语 范围	动量
连动词及从句语义关系	结果事件 条件	接续 并列	伴随 递进	时间过程 让步	除了
特殊句法结构	“的”字依存 语气依存	… 是 … 的 依存 关联词依存	连接依存 趋向动词 依存	方位词依存 介词依存	时态语态依存
特殊关系	核心成分	依次失败			

3.5.2 依存句法分析实现

按照传统的方式进行依存句法分析，主要分为两个步骤：判定和分类。判定是一个二分类问题，用于判定两个词之间是否存在依存关系；而分类则是一个多分类问题，通过依存关系判定在确定句子成分之间存在依存关系的情况下，依存关系分类给句子成分之间确定其所属的关系。传统的依存句法分析方式相对比较复杂，一般也可以采用最大生成树模型构建依存句法分析器。

基于最大生成树模型的依存句法分析大致思路：首先通过计算两个词语 A 和 B 之间的依存关系概率、词语 A 的词性与词语 B 构建依存关系概率、词语 B 的词性与词语 A 构建依存关系概率，以及词语 A 、 B 词性之间的依存关系概率，利用这些值计算词语 A 和词语 B 之间多条依存句法边，权值为上述计算的四种概率的综合；然后取边上权值最大的作为唯一的边，并加入到有向图中；最后在有向图中利用 Prim 算法构造最大生成树即可。例如，对句子“我看电影”进行依存句法分析，大致步骤如下：

(1) 分词与词性标注。对“我看电影”进行分词和词性标注，结果为“我 /rr 看 /v 电影 /n”。

(2) 确定图节点。根据图 3-10 中，可以发现所有数据节点中还包含一个“Root”节点，它是句子中核心成分指向的点，因此引入“Root”节点之后，对于“我看电影”则存在四个点，分别是“# 核心成分 #/Root”“我 /rr”“看 /v”“电影 /n”。

(3) 生成有向图。生成有向图之前，需要获得词语与词语之间的依存关系情况，而这些则通过计算类似表 3-19 中的语料库信息而得。上一步及确定图中存在四个点，则这四个点两两之间都可能存在依存关系，所以一共会构成 12 条边（值），图中四个点与点之间的关系如表 3-21 所示。

表 3-21 “# 核心成分 #/Root、我 /rr、看 /v、电影 /n” 的依存关系统计

词 语	词 语	依存关系
# 核心成分 #/Root	我 /rr	未知成分（0）
# 核心成分 #/Root	看 /v	未知成分（0）
# 核心成分 #/Root	电影 /n	未知成分（0）
我 /rr	# 核心成分 #/Root	核心成分（0.31）
我 /rr	看 /v	施者（0.22）、经验值（0.13）、受事（0.11）…
我 /rr	电影 /n	限定（0.09）、受事（0.12）、目标（0.05）…
看 /v	# 核心成分 #/Root	核心成分（0.38）
看 /v	我 /rr	连接依存（0.05）、施者（0.04）…
看 /v	电影 /n	受事（0.22）
电影 /n	# 核心成分 #/Root	核心成分（0.18）
电影 /n	我 /rr	连接依存（0.03）
电影 /n	看 /v	限定（0.29）

将表 3-21 中的依存关系统计信息生成有向图，如图 3-13 所示，图中“看”到“我”之间的边信息“连接依存（0.05）、施者（0.04）”表示为两条边的集合，括号内为边的带权值。

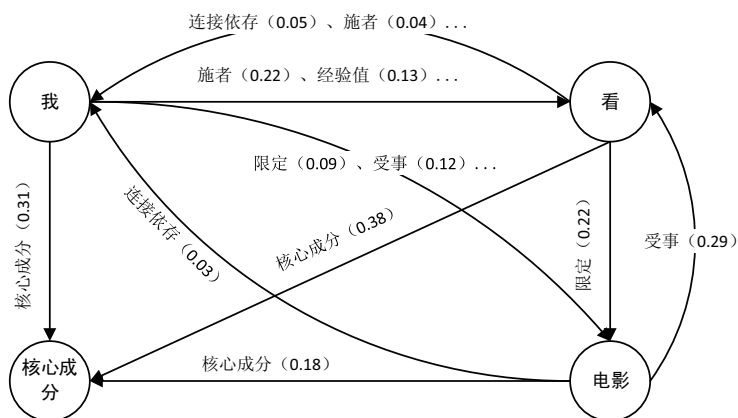


图 3-13 “# 核心成分 #/Root、我 /rr、看 /v、电影 /n” 生成的有向图

(4) 利用 Prim 算法构造最大生成树。最大生成树实质上是在图 3-13 中选择一条最佳带权路径，使得路径权值之和最大。首先，从图中点“核心成分”开始，选择指向它的最大路径权值 0.38 对应点“看”，按照 Prim 算法此刻再选择距离“核心成分”与“看”路径权值最大的点，但是由于依存句法中“核心成分”仅允许存在一项，因此不再以“核心成分”作为观察点，选择距离点“看”拥有最大路径权值 0.29 的点“电影”；其次，再对点“看”和“电影”选择指向它们的最大路径权值点 0.22 的点“我”，执行到此步已经不存在未被访问的点，则意味着上述过程完成了最大生成树的选择。因此，“# 核心成分 #/Root、我 /rr、看 /v、电影 /n”最终构成的最大生成树如图 3-14 所示。

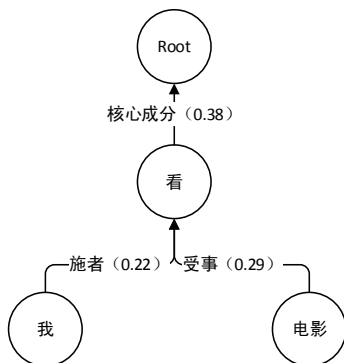


图 3-14 “# 核心成分 #/Root、我 /rr、看 /v、电影 /n” 构造的最大生成树

构造最大生成树之后，可以依据最大生成树构建依存句法关系结构图，图 3-15 所示为“我看电影”的依存句法关系。

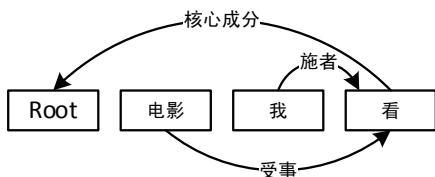


图 3-15 “我看电影”的依存句法关系

3.6 情感倾向分析

情感是用户进行搜索过程中的重要行为之一，是分析用户搜索行为的重要参考。用户的情感简单地地区分为“喜”“怒”“哀”“乐”，情感不仅仅有倾向的方向，还有在这个方向上的大小。

情感分析的目的是识别出用户在语言表达过程中心理状态的两极观点。计算机需要做的是将用户的搜索表达转换为计算机可识别的情感数值，数值大小代表着情感积极与消极程度，情感语料库是收集关于人在互联网上积极与消极的语句表达方式，通过对语料库进行分析实现情感识别，在实现过程中需要考虑 4 种特殊情况。

(1) 否定词。例如，“我喜欢编程”和“我不喜欢编程”表现为完全相反的含义，否定词的出现会让句子的含义发生很大变化。然而不仅需要判定否定词的出现，否定词的出现次数也对情感分析产生两极影响，例如，句子“我不是不喜欢编程”中的双重否定。

(2) 程度副词。对本身喜欢或不喜欢的事物再加以深刻阐述，例如，“我非常喜欢编程”“我非常不喜欢编程”，这些类似词语包括“常常”“最”“很”“几

乎”“几乎不”等。程度副词不影响情感的倾向性，但对倾向程度的影响较大。

(3) 关系连词。在复合句子中常常遇到的，包含因果关系、转折关系等，尤其在转折关系中，需要以转折后的语句作为分析标准，例如，句子“虽然他曾经不喜欢编程，但是经历了一些事情之后，他爱上了编程”。

(4) 句子类型。在正常语句的处理情况下，大多是针对陈述句，在其他句式类型中，可能表达的是另外一种情感。在疑问句中，需要通过上下文环境进行分析，例如“我喜欢编程吗？”，在感叹句中表现为另外一种加强方式，例如，“我喜欢编程！”就比陈述句“我喜欢编程”的程序倾向性更强。

通过上述 4 种特殊情况，程度副词可以通过权重控制，情感加强型副词可以通过权重控制，例如，设定权值 1.5 表示加强一半程度，情感减弱型副词取值 0.5 表示情感减弱一半的程度。否定词效果采用 $[2+(-1)n]/2$ 进行权重计算，否定词为奇数个表示越消极，偶数为积极，同程度副词达到相同的效果。

可以通过比例换算的方式判定情感倾向，词语的情感倾向通过语料库训练计算得出。每个词在积极和消极中都占有权重，对于给定的句子进行情感程度识别时，需要将句子进行分词，并结合上述 4 种特殊情况，计算每个词的消极权重和积极权重，根据词的消极权重和积极权重累计求和，计算整个句子的消极累计指数和积极累计指数，则句子的总体情感指数为两者之和，积极累计指数在总体情感指数中的占比即为句子的情感倾向，情感倾向指数越大，表明越积极，在 0.5 的时候，表示情感不积极也不消极，例如，表 3-22 所示为词语的消极指数和积极指数。

表 3-22 词语“美好”“生活”“我”“向往”的消极指数和积极指数

词语	情感消极指数	情感积极指数
美好	0.08	0.35
生活	0.45	0.51

续表

词语	情感消极指数	情感积极指数
我	0.5	0.5
向往	0.3	0.78

因此，对于计算句子“我向往美好生活”的情感指数，首先计算情感消极指数累计值为 1.33；其次计算情感积极指数累计值为 2.14；最终通过两者在情感中的占用比例计算情感指数。因此，情感识别指数为 $2.14/(1.33+2.14) \approx 0.617$ ，表示句子“我向往美好生活”带有积极的情感。

情感倾向分析也可以采用分类的方式，但是分类方式同样需要考虑上述 4 种特殊情况。

3.7 文档关键词抽取

抽取文档关键词是对网页文档进一步了解的基础。从搜索引擎角度讲，抽取文档关键词时给予不同词语、不同权重有助于理解网页本身最基本的表达含义，对于提升搜索体验也极具帮助。

3.7.1 关键词抽取概述

从网页文档分析角度讲，一般情况下可以从如下三个方面来考虑文档的关键词抽取。

(1) 网页元数据 (meta data)。某些网页会通过 meta data 指定网页关键词，例如，某新闻页面自动生成的关键词，所有的页面都有通过元数据标识关键词的可能性，如下所示。

```
<meta name="keywords" content=" 青年 , 可持续发展 , 发展 " />。
```



(2) 网页文档内容自动标注。某些内容类网站，在内容结束时大多会给予这个内容标签、标注关键词，如图 3-16 所示。

[责任编辑：PN053]

标签：政府债券 俄罗斯联邦 Anton

图 3-16 文章标注关键词示例

(3) 系统自动化抽取。关键词中最重要的，需要系统自动化地完成抽取工作。常用的自动化抽取方法是利用 TF-IDF 算法，大致思想是如果一个关键词在某网页中出现频率很高，在其他网页中很少出现，则认为它是该网页的关键词。另外一种方法则是采用词语权重（TextRank）的方法进行。也有采用语义相似度计算关键词的方法，但是考虑到性能，一般工程领域优先采用 TF-IDF 或者 TextRank 方法。

3.7.2 基于 TF-IDF 算法

TF-IDF (Term Frequency-Inverse Document Frequency) 是一种常用于检索系统的加权技术。它的原理相对比较简单，基本思想是每个字词的重要性随着它在文件中出现的次数成正比，与在其他文件中出现的次数成反比。例如，只有 A 文档中出现的某关键字，其他文档中均不存在该关键词，则认为这个关键字权值在 A 文档中相对较大，但若该关键词不仅 A 文档有，其他文档中几乎都有出现，那么它的权值会相对减小。进一步分析即如果一个单词在一篇文档中出现的频率很高，并且在其他文档集合中出现的频率很低，那么则认为该单词属于所属文档的代表词汇，与其他文档能够形成较好的区分能力。TF-IDF 是文本分类向量模型中常用的方法。

用数学公式表达 TF-IDF 中的词频 (Term Frequency, TF)，如下所示：

$$TF_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

其中， $n_{i,j}$ 是指该词在文件 d_j 中出现的次数，分母表示在文件 d_j 中所有字的出现次数之和，简单的理解即为该词在此文档中的频率。

用数学公式表达逆向文件频率（Inverse Document Frequency，IDF）如下所示：

$$IDF_i = \log \frac{|D|}{|\{j: t_i \in d_j\}|}$$

其中， $|D|$ 为资料中所有文件的总数，分母表示包含词语 t_i 的文件数目。最终第 i 篇文章的第 j 个词的 TF-IDF 权值为 $TF_{i,j} * IDF_i$ 。

结合上述 TF 及 IDF 的数学公式，TF-IDF 的计算值即为 TF 与 IDF 的积。例如，给定如表 3-23 所示三篇文档，计算词语“好朋友”的 TF-IDF 值，其中文档内容已经按照空格符分词处理。

表 3-23 计算 TF-IDF 文档示例

文档编号	文档内容
<i>A</i>	我 是 你 的 好 朋 友
<i>B</i>	今 天 天 气 真 好
<i>C</i>	我 的 好 朋 友 在 哪 呢

(1) 计算 TF。文档 *A* 中只有一个“好朋友”，但是共有词组 4 个，所以 $TF=1/4=0.25$ 。

(2) 计算 IDF。词汇“好朋友”在文档 *A* 和文档 *C* 中出现过，一共有三个文档，所以 $IDF=\log(3/2)$ 。

(3) 计算 TF-IDF，将 TF 和 IDF 相乘， $TF-IDF=TF*IDF=0.25*\log(3/2)$ 。因此“好朋友”在文档 *A* 中的权值为 0.044。

TF-IDF 并不是一个完美的权值计算方法，因为它的实现是基于“对区别文

档最有意义的词语应该是那些在文档中出现频率高，而在整个文档集合的其他文档中出现频率较低的词语”的假设条件，目的是突出重要单词，抑制次要单词，但是对于搜索引擎搜集的网页来说，在使用 TF-IDF 时，还需要考虑单词在网页中的位置信息。单词位置依然是非常关键的信息。例如，在标题中的关键词权重应当高于正文内部的权重，正文首尾段落单词的权重，应该高于正文中部文本单词权重。

3.7.3 基于 TextRank 算法

基于 TextRank 的算法思想原理借鉴于 PageRank 的网页之间相互投票的思想，把每个字当作一个网页，字与字之间也相互投票，以它们相邻的距离远近程度作为权重。下面借用 PageRank 的思想来阐述关键词的重要性提取。

每个词的权值大小来自指向这个词的所有词语的权值之和，指向关系可以视为一种词语共现关系，这种词语共现关系与 TextRank 设定的共现窗口大小有关系，共现窗口表示以某词为中心的前后 K 个词语距离内，其他词语出现的累计次数，这个次数即为共现关系的大小。通过 TextRank 算法实现最佳关键词的提取，TextRank 公式如下：

$$WS(v_i) = (1 - d) + d * \sum_{v_j \in (v_i)} \frac{w_{jk}}{\sum_{v_k \in out(v_j)} w_{jk}} WS(v_j)$$

与 PageRank 不同的是，引入了 w 的概率， w_{ji} 表示 w_j 到 w_i 的路径权重。设定词语相距越远，权重越低， d 为阻尼系数，设定值为 0.85。计算过程如图 3-17 所示。



图 3-17 基于 TextRank 计算关键词示例

(1) 首先构建短文的词网。词语与词语之间的相互作用构成一个关系网。词网关系的引入，是把每个词当作图中的一个节点。

(2) 然后利用 TextRank 算法迭代计算出每个词语的权重排序结果，该计算结果是通过不断迭代产生的。

例如，对“卖鲜花的漂亮女孩在买鲜花”分词并作词性标注后为“卖|v 鲜花|n 漂亮|a 女孩|n 在|p 买|v 鲜花|n”，保留词性标注后的名词、动词、形容词及副词，停用词及其他不相关词汇均移除。得到结果“卖 鲜花 漂亮 女孩 买 鲜花”，TextRank 算法则将利用这些词语形成一个图，每个词语作为图中的一个顶点，将词的共现关系当作图中的边。假设 TextRank 的共现窗口大小为 3，即以某个词为核心其前后相邻三个词语的出现次数。对“卖 鲜花 漂亮 女孩 买 鲜花”进行共现次数计算，得到表 3-24，表中的数值表示词的共现次数大小，例如，词语“卖”的前后三个词为“鲜花”“漂亮”“女孩”，出现次数均为 1。

表 3-24 基于 TextRank 的“卖 鲜花 漂亮 女孩 买 鲜花”的共现次数统计

	卖	鲜花	漂亮	女孩	买
卖	0	1	1	1	0
鲜花	1	0	2	2	2
漂亮	1	2	0	1	1
女孩	1	2	1	0	1
买	0	2	1	1	0

将表 3-24 转换为图中顶点之间的关系，则如图 3-18 所示。

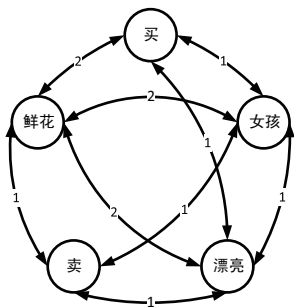


图 3-18 基于 TextRank 的“卖 鲜花 漂亮 女孩 买 鲜花”关系图

设定初始状态下，它们的权重相同，均为 1，根据 TextRank 公式进行第一次迭代计算，例如，对 $S[“卖”]$ 的计算方式如下：

$$\begin{aligned} S[“卖”] &= (1-0.85) + 0.85 * (S[“鲜花”] * 1/7 + S[“漂亮”] + S[“漂亮”]) \\ &= (1-0.85) + 0.85 * (1 * 1/7 + 1 * 1/5 + 1 * 1/5) \\ &\approx 0.611\ 428\ 571\ 428\ 571 \end{aligned}$$

然后依次计算 $S[“鲜花”]$ 、 $S[“漂亮”]$ 等，对所有词分别进行 1 次、5 次、100 次、1 000 次迭代之后，结果如表 3-25 所示。

表 3-25 “卖 鲜花 漂亮 女孩 买” 不断迭代后的权值表

	初始值	1 次迭代	5 次迭代	100 次迭代	1000 次迭代
卖	1	0.611 428 571 428 571	0.676 629 954 236 224	0.673 662 438 969 695	0.673 662 438 969 695
鲜花	1	1.405 954 058 210 88	1.410 809 639 978 74	1.405 668 364 219 31	1.405 668 364 219 31
漂亮	1	1.052 595 918 367 35	1.042 636 438 929 06	1.038 159 228 235 9	1.038 159 228 235 9
女孩	1	1.061 537 224 489 8	1.042 384 468 938 07	1.038 159 228 235 9	1.038 159 228 235 9
买	1	0.856 260 457 414 966	0.847 507 555 145 859	0.844 350 740 339 183	0.844 350 7403 391 83

通过表 3-24 可以发现，迭代计算最后自动完成收敛，完成 100 次迭代及完成 1 000 次迭代得到的结果一致。通过表 3-24 可知，权值从高到低依次为“鲜花”“漂亮”“女孩”“买”“卖”。通过表 3-25 可以发现词语“漂亮”和“女孩”在多次迭代之后，两者权重相当，这与选择的共现窗口大小有关系，例如，当选择共现窗口大小为 2，分别多次迭代后的结果如图 3-26 所示。

表 3-26 当选择共现窗口大小为 2，分别多次迭代后的结果

	初始值	1 次迭代	5 次迭代	100 次迭代	1000 次迭代
卖	1	0.532 5	0.614 516 443 508 375	0.614 840 341 040 543	0.614 840 341 040 543

续表

	初始值	1 次迭代	5 次迭代	100 次迭代	1000 次迭代
鲜花	1	1.316 178 231 099 72	1.410 809 639 978 74	1.346 969 656 425 17	1.346 969 656 425 17
漂亮	1	1.092 660 625	1.109 320 943 245 59	1.109 908 232 697 71	1.109 908 232 697 71
女孩	1	1.106 553 299 479 17	1.082 705 753 657 38	1.083 250 662 924 56	1.083 250 662 924 56
买	1	0.837 847 750 618 489	0.844 655 716 021 246	0.845 031 106 912 012	0.845 031 106 912 012

通过表 3-26 可以发现词语“漂亮”与“女孩”的权重不再相等，这也就是说共现窗口的大小也会影响词与词之间的关系分布，但是相对整体而言对词语权值大小的位置影响相对较小。一般进行 TextRank 迭代收敛之后，也会对词语权重进行归一化处理，所谓归一化是将词语的权值限定在 [0,1] 的区间，可以通过每个词语的权值除以整个词语集中最大的权值。如表 3-26 所示，以词语“鲜花”的权值作为最大权值，然后将其他词语权值与最大权值求商，得到最终结果如表 3-27 所示。

表 3-27 利用最大值对权值进行归一化处理的结果

	卖	鲜花	漂亮	女孩	买
权值	0.614 840 341 040 543	1.346 969 656 425 17	1.109 908 232 697 71	1.083 250 662 924 56	0.845 031 106 912 012
归一化权值	0.456 461 909 233 55	1.0	0.824 003 887 107 91	0.804 213 114 789 21	0.627 357 196 121 62

归一化后的权值即为文档中关键词的最后权值，进行归一化的目的是为了使得数据更具备一定范围内的可比较性。

3.8 文档句子相似度分析

搜索引擎在很多情况下都会对句子进行相似度分析，如网页标题相似度计

算、新闻去重等场景。进行文档句子相似度分析可以采用词频统计及余弦相似性（Cosine Similiarity）分析，基本思想是两个句子或文档越相似，则它们的内容（文本）也越相似，因此，可以采用 TF-IDF 对词语进行分析，然后利用余弦相似性对词语向量大小进行计算。

3.8.1 句子相似度

如表 3-28 所示给定的两个句子，计算两个句子之间的相似度。

表 3-28 句子相似度分析

句子编号	句子内容
<i>A</i>	北京未来天气晴朗、阴雨、晴朗
<i>B</i>	上海未来天气阴雨、多云、晴朗

针对表 3-28 中的句子内容，需要分步进行相似度计算，步骤如下：

（1）分词处理。对表 3-28 中的句子进行分词处理，移除标点符号。处理结果如表 3-29 所示。

表 3-29 示例句子分词结果

句子编号	句子内容
<i>A</i>	北京 未来 天气 晴朗 阴雨 晴朗
<i>B</i>	上海 未来 天气 阴雨 多云 晴朗

（2）计算词频。计算分词结果中每个句子中词语的频率。对于其他句子中存在而本句中不存在的词语，设定词频值为 0，因此对句子 *A*、*B* 的词频计算结果如表 3-30 所示。

表 3-30 句子词频计算结果

句子编号	词频
<i>A</i>	北京 (1) 未来 (1) 天气 (1) 晴朗 (2) 阴雨 (1) 多云 (0)
<i>B</i>	上海 (1) 未来 (1) 天气 (1) 晴朗 (1) 阴雨 (1) 多云 (1)

（3）构建词频向量。针对句子 *A*，产生的词频向量为 [1,1,1,2,1,0]；针对句子 *B*，

产生的词频向量为 [1,1,1,1,1,1]。到此步则已经将问题转变为求向量的相似度问题。

(4) 向量相似度计算。向量相似度可以转变为两条向量夹角的大小，如图 3-19 所示。如果向量夹角为 0° ，则表示向量的方向相同、线段重合；如果向量夹角为 90° ，则表示两个向量完全不相似；如果向量夹角为 180° ，则表示方向完全相反。因此，可以通过夹角的大小来判断向量的相似度，其中夹角越小，则相似度越高。

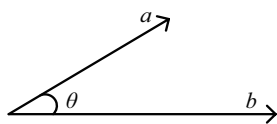


图 3-19 向量夹角示例

对于求解两个向量之间的夹角 θ ，可以通过 $\cos \theta$ 的值体现。结合图 3-19， $\cos \theta$ 可以根据余弦定理公式求解，公式如下：

$$\cos \theta = \frac{a^2 + b^2 - c^2}{2ab}$$

将向量 a 定义为 $[x_1, y_1]$ ，向量 b 定义为 $[x_2, y_2]$ ，则余弦定理公式可推导为如下所示，余弦定理不仅仅对二维向量有效，对多维向量也适用，方法类似。

$$\cos \theta = \frac{x_1 x_2 + y_1 y_2}{\sqrt{x_1^2 + y_1^2} * \sqrt{x_2^2 + y_2^2}}$$

因此根据余弦定理，计算向量 [1,1,1,2,1,0] 与向量 [1,1,1,1,1,1] 的角度，如下所示：

$$\cos \theta = \frac{1*1+1*1+1*1+2*1+1*1+0*1}{\sqrt{1^2+1^2+1^2+2^2+1^2+0^2} * \sqrt{1^2+1^2+1^2+1^2+1^2+1^2}} = \frac{6}{\sqrt{8} * \sqrt{6}}$$

综上所述，向量 [1,1,1,2,1,0] 与向量 [1,1,1,1,1,1] 的 $\cos \theta$ 值约为 0.866，即可以认为句子 A、B 之间的相似度为 86.6%。

3.8.2 文档相似度

文档相似度的计算过程同句子相似度类似，但并不是将文档中的所有句子拆分，然后对句子进行相似度比较。基于 3.7 节中对关键词抽取方法介绍，对于需要比较的两篇文档，可以通过提取各自关键词，利用关键词进行词频统计，以及再通过余弦相似性计算。

但是如果仅仅采用关键词的方式只能粗略进行相似度估算，因为其没有结合关键词出现的位置进行综合分析，因此另外一种方法是采用 w-shingling 算法。w-shingling 算法的核心思想是将两篇文档的相似性转变为两篇文档集合的相似性问题。w-shingling 算法中的集合是文档内容中有序序列，采用 N-Gram 的方式提取句子中的有序词组，如表 3-31 所示给定的两篇示例文档 *A*、*B*。

表 3-31 文档相似度给定的示例文档

文档编号	内 容
<i>A</i>	我去吃饭。不去吃饭。
<i>B</i>	我去吃饭。不去理发。

在对文档 *A* 和文档 *B* 去除无关标点符号之后，采用 Tri-Gram（三元组）的方式分别对文档 *A*、*B* 提取有序序列，提取结果如表 3-32 所示。

表 3-32 对示例文档进行 Tri-Gram 处理

文档编号	Tri-Gram 集合
<i>A</i>	{ 我去吃、去吃饭、不去吃、去吃饭 }
<i>B</i>	{ 我去吃、去吃饭、不去理、去理发 }

分别对两个文档中的 Tri-Gram 集合过滤重复序列，得到去重后的序列结果如表 3-33 所示。

表 3-33 对文档的 Tri-Gram 分词结果进行去重过滤

文档编号	Tri-Gram 集合
<i>A</i>	{ 我去吃、去吃饭、不去吃 }
<i>B</i>	{ 我去吃、去吃饭、不去理、去理发 }

w-shingling 算法认为两个文档的相似度为文档子集的交集个数与并集个数之比。如下公式所示：

$$r(A,B) = \frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|}$$

根据公式可知，文档之间共同的有序词组越多则文档相似度越高。根据示例， $r(A,B)=2/7 \approx 0.286$ 。w-shingling 算法与关键词的不同点在于引入的有序词组序列，这样的序列将文档中的词汇出现位置进行综合考虑。如果此处不采用 N-Gram 提取有序序列，而采用中文分词对文档词语进行拆分，然后对文档求交集个数与并集个数之比，则也会丢失序列关系。

3.9 文档核心句抽取

文档核心句又称为文档中心句。提取文档核心句，是为提升用户体验而设计的。在用户搜索过程中，难免会遇到用户搜索的关键词只在标题中出现，而不在内容中出现的情况，此刻，若系统能够自动将文档的核心句提取出并展示给用户，使得用户在最短时间内大致能够了解文章需要表达的内容，将会给用户搜索体验带来提升。

从技术角度来讲，文档核心句是从文档中自动抽取关键句。在 3.7 节中对文本关键词的抽取有详细介绍。从工程应用角度存在两种方法可以做到核心句的抽取，第一种也是最容易想到的方法是基于 3.7 节已经获取的所有关键字在文档中的权重，利用均值处理方式计算某一句包含的关键字平均权重最大，此种方法最简单，但仅在较短文章中有较好的效果，随着文本内容容量的增加，效果将会不断衰减。

为了使对任何长文本和短文本的处理达到较好的一致效果，借助 3.7 节关

于 TextRank 算法的思想将词语权值的计算演进为句子权值的计算，这种计算方式也可以达到很好的效果，借鉴 TextRank 公式，句子权值公式如下所示：

$$TS(V_i) = (1 - d) + d * \sum_{v_j \in (v_i)} \frac{T_{ji}}{\sum_{v_k \in \text{out}(v_j)} T_{jk}} TS(V_j)$$

在关键词计算中以词语作为单位进行计算，在文档摘要中则以句子为单位。 T_{ji} 则表示两个句子的关系紧密程度，其余 $TS(V_j)$ 表示上次迭代的第 j 个句子权值。计算两个句子关系紧密程度，可以参考 3.7 节中介绍的以 TF-IDF 及余弦相似性进行相似度计算，或者通过句子关键词共有数量确定，两者在本质上是同样的。

通过下面的示例理解句子权值的计算方法，例如，对下面一段内容分析其中的关键语句。

中国对南海及南沙群岛拥有无可争辩的主权。中国在南海的主权和相关权利是在长期的历史过程中形成的，为历代中国政府所坚持。中方在自己的领土上开展建设，是主权范围内的事，不针对、不影响任何国家。各国也依国际法在南海享有航行和飞越自由。南海的航行自由没有问题，这也是铁的事实。

(1) 进行断句预处理。将段落中的文本按照句号、感叹号等分隔符切分为单个独立句子。表 3-34 所示为断句处理后的语句，因此文章的核心句也将会从表 3-34 所示的 5 个句子中产生。

表 3-34 将文档切分为单个独立句子示例

句子编号	句 子
A	中国对南海及南沙群岛拥有无可争辩的主权
B	中国在南海的主权和相关权利是在长期的历史过程中形成的，为历代中国政府所坚持
C	中方在自己的领土上开展建设，是主权范围内的事，不针对、不影响任何国家
D	各国也依国际法在南海享有航行和飞越自由
E	南海的航行自由没有问题，这也是铁的事实

(2) 对每个句子进行分词及词性标注处理，并过滤掉停用词及部分词性无关词汇，再利用 TextRank 计算每个句子中词语的权重，并筛选出每个句子中词语权值由高到低排序的前三个关键词，选取后的关键词如表 3-35 所示。

表 3-35 计算每一个句子中的关键词结果

句子编号	句 子	关键词
<i>A</i>	中国对南海及南沙群岛拥有无可争辩的主权	南海、南沙群岛、主权
<i>B</i>	中国在南海的主权和相关权利是在长期的历史过程中形成的，为历代中国政府所坚持	中国、南海、历代
<i>C</i>	中方在自己的领土上开展建设，是主权范围内的事，不针对、不影响任何国家	领土、主权、建设
<i>D</i>	各国也依国际法在南海享有航行和飞越自由	国际法、航行、南海
<i>E</i>	南海的航行自由没有问题，这也是铁的事实	南海、航行、自由

(3) 通过关键词计算句子之间的关系紧密程度，以相同的关键词数量作为紧密值。例如，句子 *A* 和句子 *B* 中共有关键词“南海”，则紧密值为 1，最终句子之间的关系紧密程度如表 3-36 所示。

表 3-36 句子之间的关系紧密程度

	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
<i>A</i>	0	1（南海）	1（主权）	1（南海）	1（南海）
<i>B</i>	1（南海）	0	0	1（南海）	1（南海）
<i>C</i>	1（主权）	0	0	0	0
<i>D</i>	1（南海）	1（南海）	0	0	2（南海、航行）
<i>E</i>	1（南海）	1（南海）	0	2（南海、航行）	0

按照与 TextRank 计算采用同样的方法，句子之间相互“投票”，将句子 *A*、*B*、*C*、*D*、*E* 的初始权值设定为 1，不断进行迭代计算并将结果归一化处理，如表 3-37 所示。

表 3-37 对句子关系做累计迭代结果与归一化处理

	初始值	1 次迭代	5 次迭代	100 次迭代	1 000 次迭代
<i>A</i>	1	1.708 333 333 333 33	1.309 651 984 175 76	1.275 479 715 294	1.275 479 715 294

续表

	初始值	1 次迭代	5 次迭代	100 次迭代	1 000 次迭代
<i>B</i>	1	0.938 020 833 333 33	0.949 882 313 100 90	0.926 209 582 768 04	0.926 209 582 768 04
<i>C</i>	1	0.513 020 833 333 33	0.428 301 046 637 35	0.421 039 439 499 97	0.421 039 439 499 97
<i>D</i>	1	1.203 793 402 777 78	1.218 138 647 001 1	1.188 635 631 218 99	1.188 635 631 218 99
<i>E</i>	1	1.290 405 598 958 33	1.215 143 293 658 07	1.188 635 631 218 99	1.188 635 631 218 99

根据表 3-37 可知, 最终确定句子核心句为第一句“中国对南海及南沙群岛拥有无可争辩的主权”, 最不重要的句子是“中方在自己的领土上开展建设, 是主权范围内的事, 不针对、不影响任何国家”。从上面的例子中也可以看出其实文档中的核心句, 并不一定是最长的句子。不难看出基于 TextRank 算法衍生的句子权重算法将句子与句子之间的紧密关系以共同拥有的关键词作为衡量, 从而将紧密程度转变为句子之间的相互影响力, 最终通过句子之间不断迭代投票的方式选出核心句, 上述过程中对句子之间的紧密程度也可以采用句子相似度的方法。

3.10 聚类分类

文本的聚类分类是文本处理过程中非常重要的环节。聚类是将某些特征相同的文本, 归纳为同一类簇, 从机器学习的角度而言, 聚类是一种无指导性的学习, 在预先不知道分类标签的情况下自发形成的信息聚集, 是“物以类聚”的数据体现; 而分类是将文本划分为某一特定已知类型, 是一种指导性学习方法, 在进行分类之前会有样本训练的过程, 而样本训练的数据集事先已经知道其分类标签, 而分类过程也是依据样本数据中的已知分类标签对未知分类数据进行分类标签标记。

3.10.1 文本分类

文本分类在搜索引擎中属于必备语言处理模块，每篇文章都由成百上千个词语组成，可以当作一个向量集 $W=(w_1, w_2, w_3, \dots, w_n)$ ，其中 w_i 即表示其中第 i 个词语。文章的分类也可以视为一个分类标记集合 $C=(c_1, c_2, c_3, \dots, c_m)$ 。在进行特征学习之前，词 w_i 与 c_j 的关系不是确定值，因此需要计算 $P(C|W)$ ，也就是在 w_i 出现的情况下，文本是文本分类 C 的概率，可根据贝叶斯计算，公式如下：

$$P(C|W) = \frac{P(W|C) * P(C)}{P(W)}$$

从文本分类的角度理解贝叶斯公式即为：在 w_i 词出现的情况下是否是文本类别 c_j 取决于在文本分类 c_j 情况下 w_i 出现的概率，以及 w_i 在所有词中出现的概率。 $P(W)$ 的意义在于如果这个词在所有文档中出现，那么用 w_i 去判定是否是 c_j 的概率越低，越不具备代表性。

朴素贝叶斯是一种有监督的学习方式，可以利用伯努利模型（Bernoulli Model）以文件为粒度进行文本分类。例如，表 3-38 所示的文档中，给定文本训练数据集合，并已知训练集中文档分类的类型。

表 3-38 文档内容与文档类型示例

文档编号	文档内容	文档分类的类型
doc1	国家 GDP	财经
doc2	国防 保卫 安全	军事
doc3	银行 利率	财经
doc4	国家 预算 减少	财经

基于表 3-38 示例，现给定判定样本“国家 国防 安全”，确定其文档分类的类型。即给定 $W=(\text{国家}, \text{国防}, \text{安全})$ ， $C=(\text{财经}, \text{军事})$ ，从直观角度几乎可以预测属于军事，但是对于计算机系统，需要模型、量化值得出准确可能性的概率值。

根据伯努利模型原理，定义：



$$P(C_j) = \frac{T(C_j)}{T}$$

其中， T 表示训练集合的文档总数，这里 $T=4$ ， $T(C_j)$ 表示分类 C_j 出现的文档数量。

$$P(w_i|c_j) = \frac{G(w_i|c_j) + 0.1}{G(c_j)}$$

其中， $G(w_i|c_j)$ 表示在分类 c_j 下拥有单词 w_i 的文档数量，0.1 是防止分子为 0 的情况。 $G(c_j)$ 表示分类 c_j 下拥有的文档数。 $c_{\text{财经}}$ 共 3 个文档， $c_{\text{军事}}$ 共 1 个文档，文档总数 $T=4$ ，因此财经占有概率 $P(c_{\text{财经}})$ 与军事 $P(c_{\text{军事}})$ 占有概率分别如下：

$$P(c_{\text{财经}}) = 3/4 = 0.75$$

$$P(c_{\text{军事}}) = 1/4 = 0.25$$

各个词汇在财经和军事分类下的条件概率计算如下：

$$P(w_{\text{国家}}|c_{\text{财经}}) = (2+0.1)/3 = 0.7$$

$$P(w_{\text{GD}}|c_{\text{财经}}) = P(w_{\text{银行}}|c_{\text{财经}}) = P(w_{\text{利率}}|c_{\text{财经}}) = P(w_{\text{预算}}|c_{\text{财经}}) = P(w_{\text{减少}}|c_{\text{财经}}) = (1+0.1)/3 \approx 0.37$$

$$P(w_{\text{国防}}|c_{\text{财经}}) = P(w_{\text{保卫}}|c_{\text{财经}}) = P(w_{\text{安全}}|c_{\text{财经}}) = 0.1/3 = 0.03$$

$$P(w_{\text{国防}}|c_{\text{军事}}) = P(w_{\text{保卫}}|c_{\text{军事}}) = P(w_{\text{安全}}|c_{\text{军事}}) = (1+0.1)/1 = 1.1$$

$$P(w_{\text{GD}}|c_{\text{军事}}) = P(w_{\text{国家}}|c_{\text{军事}}) = P(w_{\text{银行}}|c_{\text{军事}}) = P(w_{\text{利率}}|c_{\text{军事}}) = P(w_{\text{预算}}|c_{\text{军事}}) = P(w_{\text{减少}}|c_{\text{军事}}) = 0.1/1 = 0.1$$

$$P(w_{\text{国家}}) = 3/4 = 0.75$$

$$P(w_{\text{GD}}) = P(w_{\text{国防}}) = P(w_{\text{保卫}}) = P(w_{\text{安全}}) = P(w_{\text{银行}}) = P(w_{\text{利率}}) = P(w_{\text{预算}}) = P(w_{\text{减少}}) = 1/4 = 0.25$$

由上述概率可以推理出各个词汇属于财经或者军事的概率，如下：

$$P(c_{\text{财经}}|w_{\text{国家}}) = 0.7 * 0.75 / 0.75 = 0.6$$

$$P(c_{\text{财经}}|w_{\text{国防}}) = 0.03 * 0.75 / 0.25 = 0.09$$

$$P(c_{\text{财经}}|w_{\text{安全}}) = 0.03 * 0.75 / 0.25 = 0.09$$

$$P(c_{\text{军事}}|w_{\text{国家}})=0.1*0.25/0.75=0.03$$

$$P(c_{\text{军事}}|w_{\text{国防}})=1.1*0.25/0.25=1.1$$

$$P(c_{\text{军事}}|w_{\text{安全}})=1.1*0.25/0.25=1.1$$

上述计算过程是通过语料库的离线训练得到的，每个词语对应一组分类向量集合，包含词语在分类向量中的权重。从搜索引擎角度看，分类的性能消耗要做得足够小，尤其在给用户搜索词进行快速分类时。

$$\text{因此, } P_{\text{财经}} = P(c_{\text{财经}}|w_{\text{国家}}) * P(c_{\text{财经}}|w_{\text{国防}}) * P(c_{\text{财经}}|w_{\text{安全}}) = 0.6*0.09*0.09=0.004\ 86$$

$$P_{\text{军事}} = P(c_{\text{军事}}|w_{\text{国家}}) * P(c_{\text{军事}}|w_{\text{国防}}) * P(c_{\text{军事}}|w_{\text{安全}}) = 0.03*1.1*1.1=0.036\ 3$$

显然， $P_{\text{军事}}$ 大于 $P_{\text{财经}}$ ，且 $0.036\ 3/(0.036\ 3+0.004\ 86)=0.88$ 。结论即“国防 安全”有 88% 的可能性属于军事类型，仅有 12% 的可能性属于财经类型。

通过上述计算过程，可以归纳朴素贝叶斯大致分为数据准备、分类器训练及分类识别三个阶段。

(1) 数据准备。语料库的准备工作阶段，这个阶段的任务是为朴素贝叶斯分类做必要的准备，主要工作是根据具体情况确定特征属性，并对每个特征属性进行适当划分，然后由人工对一部分待分类项进行分类，形成训练样本集合。这一阶段的输入是所有待分类数据，输出是特征属性和训练样本。这一阶段是整个朴素贝叶斯分类中唯一需要人工完成的阶段，其质量对整个过程将有重要影响，分类器的质量很大程度上是由特征属性、特征属性划分及训练样本质量决定的。

(2) 分类器训练。这个阶段的任务是生成分类器，主要工作是计算每个类别在训练样本中的出现频率及每个特征属性划分对每个类别的条件概率估计，并将结果记录。其输入是特征属性和训练样本，输出是分类器。这一阶段是机械性阶段，根据前面讨论的公式可以由程序自动计算完成。

(3) 分类识别。这个阶段的任务是使用分类器对待分类项进行分类，其输入是分类器和待分类项，输出是待分类项与类别的映射关系。这一阶段也是机械性阶段，由程序完成。

分类的方法林林总总，主要以文档为粒度进行分析，也可以采用词频的方式进行分析，还可以采用 TF-IDF，然后加权求值方式。

在分类结果中，常常需要将数值归一化，尤其是对分类器训练的结果，归一化的目的是让每个词在不同分类上的权重对应到区间 (0,1)，以使得语料库中的词在相同分类的权值相加变得更有可比性。

归一化是一种数据预处理方法，就是要把需要处理的数据经过特定算法处理后，限制值区间在一定范围内，这样做不仅是为了后面数据处理的方便，也是为了保证程序运行时加快收敛。

归一化常用的三种方法包括 $\text{atan}(x)*2/\text{PI}$ 、 $\log_{10}^{(x+1)/(\max(x)+1)}$ 、 $\frac{\text{atan}(\log_2^{x-k+1}) * 2}{\text{PI}}$ 等方式，对于词在每个分类中的权值按照这三种方式进行归一化后，权值分布如图 3-20、图 3-21 和图 3-22 所示。

(1) 采用 $\text{atan}(x)*2/\text{PI}$ 方式。

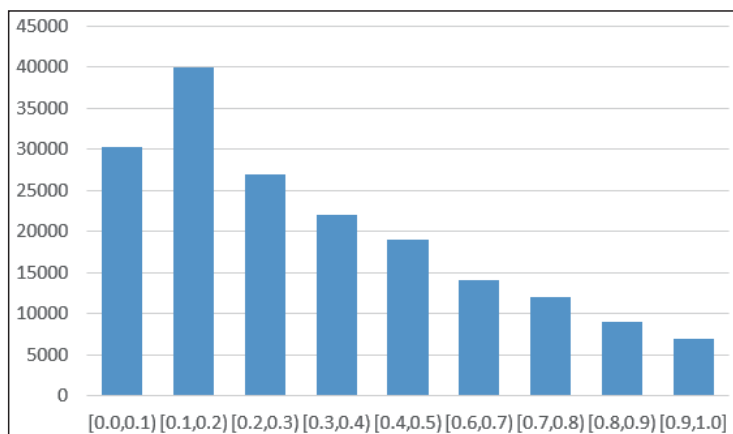


图 3-20 归一化采用 $\text{atan}(x)*2/\text{PI}$ 的词分类权值分布

(2) 采用 $\log_{10}^{(x+1)/(\max(x)+1)}$ 方式。

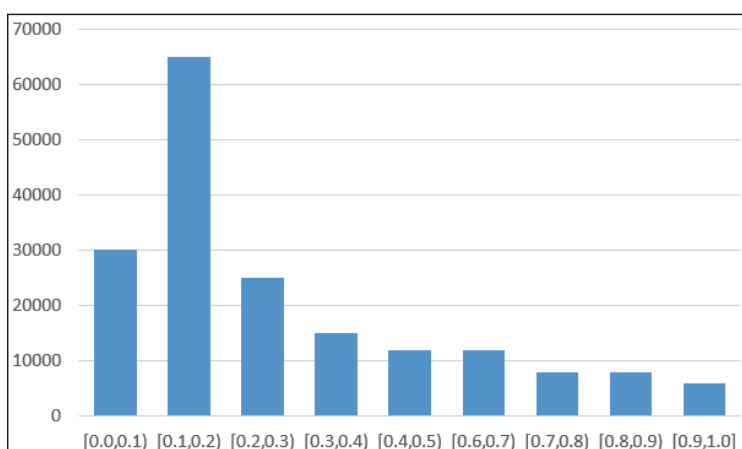


图 3-21 归一化采用 $\log_{10}^{(x+1)/(\max(x)+1)}$ 的词分类权值分布

(3) 采用 $\frac{\text{atan}(\log_2^{x-k+1}) * 2}{\text{PI}}$ 方式。

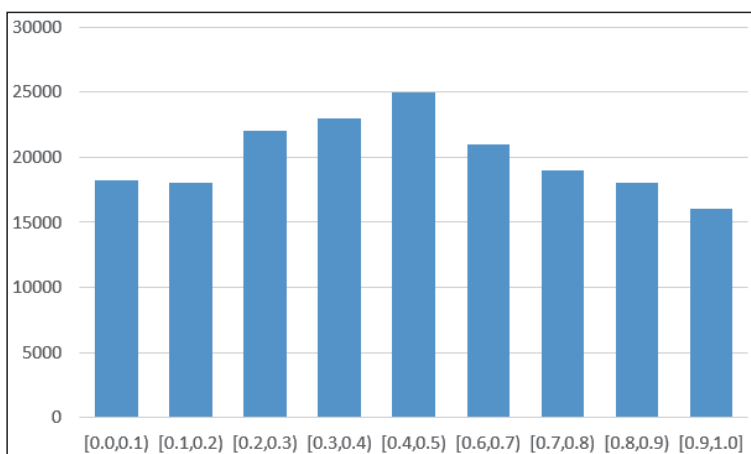


图 3-22 归一化采用 $\frac{\text{atan}(\log_2^{x-k+1}) * 2}{\text{PI}}$ 的词分类权值分布

上述第三种方法更适合针对当前数据的归一化，原因在于数据分布更加均匀。当然其中的 k 值，是通过不断调整出来的值，针对不同的语料库不一样，通过调整，可以使得每个单词元组在从分类向量向上叠加或相乘的过程中，使



得它们的权值大小具备可比性，否则会导致权值差距过大。但期望保持这样的相对大小关系，却不想保留这样的差值，差值过大会影响语料库相应词汇在某分类上的权重。因此并不是任选一种归一化方法都可以达到目的，归一化方法对系统的结果有非常重大的影响，不是简单地将数值划分到一个区间，而是需要不断调整归一化后的值分布，调整的目标是尽可能让归一化后的值均匀分布。

3.10.2 文本聚类

聚类分析是一种无监督的机器学习方法，聚类和分类不同的是，它不需要通过语料库训练，更不需要早期的人工标注类型，具备较高的灵活性和极高的自动化处理能力。

聚类分析的方法包括自上而下的分析和自下而上的分析两种方式。

(1) 自上而下的分析方法。先把所有样本视为一个聚类，然后不断地从这个大的聚类中分离出更多小聚类，直到不能再继续分离为止。

(2) 自下而上的分析方法。将局部样本形成一个小聚类，然后通过不断聚合小聚类，最终形成几个大的聚类。

文本聚类可以帮助系统进行强大的后台分析，尤其在分析用户行为的时候，用于确定相似用户。相似用户分析对于了解搜索引擎用户分布及行为预测有相当程度的帮助，对广告投放依然如此，也有研究员提出对搜索引擎返回的页面进行聚类，使得用户可以迅速定位到自己需要的信息。

文本聚类算法中，最直接的是 K-Means 算法，K-Means 算法也是最为经典、最为基本的数据挖掘算法，它是一种基于自下而上的聚类分析方法。基本概念就是：空间中有 N 个点，初始选择 K 个点，作为中心聚类点，将 N 个点分别

与 K 个点计算距离，选择最近的作为自己的中心点，不断地更新中心聚集点。

K-Means 解决的问题如图 3-23 和图 3-24 所示。

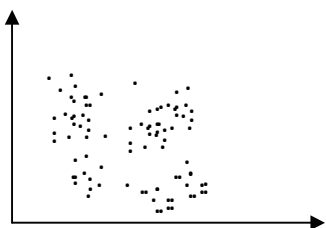


图 3-23 需要聚类的数据分布情况

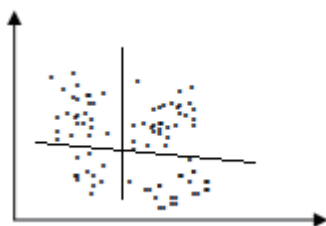


图 3-24 通过 K-Means 聚类后效果

K-Means 算法能够将图 3-23 所示的零散数据分布进行划分，并切分为严格的几个聚类，聚类结果如图 3-24 所示，以达到“物以类聚”的效果。K-Means 的 K 表示聚类的个数，在一开始即为一个固定值。下面通过示例分析 K-Means 的工作原理，对于指定文本如下所示，需要将下面的语句进行聚类分析。

美联储将举行货币政策会议，欧洲、中国等地对于全球金融市场再度挑起战火

光大银行推出公私客户全息电商金融服务平台

现在拉齐奥距离第一位差了 2 分，拉齐奥也应该思考争夺意甲冠军

奇酷手机旗舰版双 4G 版发售，青春版降价 200

资本市场服务实体经济还体现在对国家重大战略的支持上

高通定义的移动范畴已经远远超越手机产业的上下游

中国女足凭借一平一胜的战绩获得本届邀请赛冠军

移动回应手机流量事件：或因后台自动更新

卡帅加盟新东家不忘恒大：我原本能带队夺中超冠军

中国大力推动足球革命外媒：球迷“忍了很久”

中移动宣布终端高管离职，终端公司终将渠道化

很明显上述语句主要属于三大类：财经、体育和互联网信息。



(1) 通过分词，过滤停用词。

美联储将举行货币政策会议，欧洲、中国等地对于全球金融市场再度挑起战火

光大银行推出公私客户全息电商金融服务平台

现在拉齐奥距离第一位差了 2 分，拉齐奥也应该思考争夺意甲冠军。

奇酷手机旗舰版双 4G 版发售，青春版降价 200

资本市场服务实体经济还体现在对国家重大战略的支持上

高通定义的移动范畴已经远远超越手机产业的上下游

中国女足凭借一平一胜的战绩获得本届邀请赛冠军

移动回应手机流量事件：或因后台自动更新

卡帅加盟新东家不忘恒大：我原本能带队夺中超冠军

中国大力推动足球革命外媒：球迷“忍了很久”

中移动宣布终端高管离职，终端公司终将渠道化

(2) 设定 K 值，计算距离。在通常的 K-Means 计算中，距离一般是二维坐标中的坐标距离，即两个坐标点 $A(x_1, y_1)$, $B(x_2, y_2)$ ，则距离公式为：

$$|AB| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

但是对于句子的距离，则约定为句子的相似度，可参考 3.7 节中介绍的以 TF-IDF 及余弦相似性进行相似度计算。假定需要设定的聚类数量 k 为 3。

(3) 不断迭代计算。在迭代计算之前，需要假定初始状态下三个聚类的中心点位置，一般是随机句子中的三句，分别为聚类 k_1 、 k_2 、 k_3 的中心。但是以实际经验来讲，这种方式在某些情况下效果较差，应当选择彼此距离较远的三个点。以示例中句子为例找相似度较远的三个句子。其实也不难理解，相似度越远，说明它们属于不同聚类的可能性越大，有助于数据收敛，并减少迭代次数。

选择好三个初始的类簇中心点后，将其他句子分别与三个初始类簇中心点计算相似度，例如第 m 个句子， S_m 分别计算其与聚类 k_1 、 k_2 、 k_3 的中心句子相似度 d_{m1} 、 d_{m2} 、 d_{m3} ，若 d_{m1} 值最小，则说明在本次迭代中 S_m 属于聚类 k_1 ，依次类推。完成一次迭代之后，需要重新确定聚类 k_1 、 k_2 、 k_3 的中心句子。确定一个聚类的中心句子的方式是计算聚类的每个点到中心点平均距离最短的点，确定完类簇中心句子之后，再次迭代计算每个句子和类簇中心距离，直至数据收敛。在有的时候数据一直难于收敛，处于少量波动状态，则可以通过设定最大迭代次数进行约束，这样的可能性一般存在于确定的聚类个数不够，导致某些数据无法确定属于哪个聚类，另外可能是中间状态数据过多，某一数据处于两个聚类之间或者数据较为均匀的分布，随着中心点的改变，在边界范围波动。

(4) 通过 K-Means 不断迭代示例中的句子，最终聚类结果如表 3-39 所示。

表 3-39 文本聚类结果示例

聚 类	句 子
第一聚类	现在拉齐奥距离第一位差了 2 分，拉齐奥也应该思考争夺意甲冠军 中国女足凭借一平一胜的战绩获得本届邀请赛冠军 卡帅加盟新东家不忘恒大：我原本能带队夺中超冠军 中国大力推动足球革命外媒：球迷“忍了很久”
第二聚类	美联储将举行货币政策会议，欧洲、中国等地对于全球金融市场再度挑起战火 光大银行推出公私客户全息电商金融服务平台 资本市场服务实体经济还体现在对国家重大战略的支持上
第三聚类	奇酷手机旗舰版双 4G 版发售，青春版降价 200 高通定义的移动范畴已经远远超越手机产业的上下游 移动回应手机流量事件：或因后台自动更新 中移动宣布终端高管离职，终端公司终将渠道化

通过 K-Means 有效地将示例中文本进行聚类分析。虽然 K-Means 非常简单、快速地完成了聚类工作，但凭借笔者多年的经验，它并不非常完美，有两个非常明显的缺陷：

(1) 对异常值、摇摆值比较敏感，导致收敛变慢。



(2) 提前确定 K 值，有时难免不知道数据可能存在的聚类个数，在笔者开发过程中常常是以经验确定的值，经验来自对数据分布的感觉。

使用 K-Means 做聚类，需要对数据有一定理解。尤其是在大数据时代的今天。在解决异常值和摇摆值的问题上，不必过分担心。原因是在数亿级以上的数据量上，存在少量数据处于中间状态，对系统的影响非常小，相对其优势来讲，可以接受。

对于提前确定 K 值问题，也有较好的办法，通过多次在小数据集实验不断调整 K 值。在具体应用的数据校验准确率中，通过不同的 k 值查看准确率效果，如图 3-25 所示，通过大量实验确定 k 值为 56 时准确率最高。

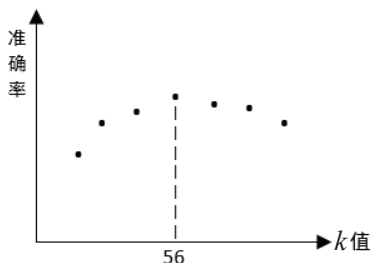


图 3-25 K-Means 中调整 K 值与准确率关系

3.11 语种检测

随着 Web 数量呈极速增长的态势且 Web 网页中各类语言之间相互涵盖，对于网页进行语种检测 (Language-Detection) 已经成为搜索引擎理解网页内容的第一步。语种检测是自然语言处理中和分词技术一样基础的技术，搜索引擎需要确保抓取的网页语言类型，方能进行分词处理等，才能更好地为全球各地的网民服务，不仅仅在抓取时，在用户进行搜索过程中，也需要检测用户输入的语种类型，以根据语种进行信息筛选最适合用户的信息。

语种检测最简单的理论方式是通过搜集各类语言的字典，计算每个字是某种语言的概率值，然后通过句子中的所有字的语种概率值，确定句子最大可能的语种。例如，对文本“永続的な友情”计算其可能所属的语种，如表 3-40 所示，表中的值为文字与语种的可能性参考值，值越高则越可能属于某个语种。

表 3-40 “永続的な友情”语言检测示例

字	简体中文	繁体中文	日 文
永	1	1	1
続	0	1	1
的	1	1	1
な	0	0	1
友	1	1	1
情	1	1	1
合计	4	5	6

因此根据表 3-40，可以确定“永続的な友情”属于日文的可能性最大，但是在项目的实际运作中，这种方式的可行性较差，原因在于：

(1) 全球共有 5 000 多种语言，常用的 50 种左右。

(2) 语言变化性强，尤其是以字母作为语言的国家，例如，英语中词有很多简写，词性变化，词也有合并的情况，各式各样的变化，使得使用字典不一定能够解决问题。

(3) 存在看似是一类文字，但实质是两类不同文字。例如，词语“溺死”和“水死”都是汉语中表达的一个含义，但是“溺死”是中文的概念，而“水死”是日文的概念。

基于上述原因，可以采用 N-Gram 的语言模型，并结合 TF-IDF 进行语种检测。

(1) N-Gram 分词。对于任何语言的语料库文本进行 N-Gram 模型分析，N 取 3，即 Tri-Gram，同时也将 Uni-Gram 和 Bi-Gram 依次取出。例如，对“水木清

华”进行 Tri-Gram、Bi-Gram、Uni-Gram 分词，取出内容分别为“水”“木”“清”“华”“水木”“木清”“清华”“水木清”“木清华”。

(2) TF-IDF 权值计算。在每一个语言的语料库内，计算每一个词的 TF-IDF 值。

(3) 语种检测。例如，检测文本“检索技术”的语种，对“检索技术”同第一步相同的方式拆分出 Tri-Gram、Bi-Gram、Uni-Gram：“检”、“索”、“技”、“术”、“检索”、“索技”、“技术”、“检索技”、“索技术”，将分成的元组在各类语言的语料库中，寻找已经计算好的 TF-IDF 值，结果如表 3-41 所示。

表 3-41 基于 N-Gram 及 TF-IDF 语言检测模型结果示例

字	简体中文	繁体中文	日 文
检	0.01	0.13	0.27
索	0.42	0.47	0.45
技	0.53	0.41	0.39
术	0.08	0.32	0.35
检索	0.01	0.14	0.34
索技	0.36	0.40	0.37
技术	0.01	0.39	0.33
检索技	0	0.05	0.29
索技术	0	0.05	0.29
合计	1.42	2.36	3.08

从表 3-41 中可以得出“检索技术”是日文的可能性比较高。利用 N-Gram 语言模型，将语言的连续性引入检测，而不仅仅是单词的检测，将有助于提高检测的准确率。

语种的识别，也可以采用分类的方式解决，还可以通过先进行语系分析、再通过每个语系下的各个语种在字符集范围做对比进行语言语种识别。例如，在 UTF-8 下，中文字符集编码在 u4e00~u9fa5 区间，日文编码则在 u0800~u4e00 区间等。

3.12 本章小结

中文分词、词性分析、情感分析、关键词抽取、文档核心句抽取、聚类分类等都是搜索引擎中最基础的文本挖掘和处理方式，它们也是自然语言处理中的典型算法。不仅如此，自然语言处理作为计算机科学领域与人工智能领域中的一个重要方向，它是搜索引擎与用户进行交互的窗口，它的发展已经日趋成熟，面对互联网中无穷无尽的数据，搜索引擎以自然语言理解的方式，不断将非结构化的数据转变为结构化可理解的数据信息，服务于广大网民。

第 4 章 构建大数据存储引擎

在海量数据面前，数据存储已经变得不那么简单，丰富多样的数据，使得无法将所有数据完全结构化。从搜索引擎角度看，互联网上的网页数据在不断地增长和更新，采用传统的数据存储方式，已经不能够为数据的快速分析和搜索提供支持，搜索引擎也在通过各种存储方式解决因为数据存储导致的性能问题。在海量数据下，存在两个挑战：

（1）快速存取。庞大的数据被分散存储在各大分布式集群的服务器中，能够进行快速的数据存和取，将有助于提高单位时间内数据的处理速率。

（2）IO 性能瓶颈。磁盘的读 / 写，在分布式环境中也是非常重要的因素，如果全部采用 SSD 固态硬盘虽然可以解决 IO 缓慢问题，但是高额的硬件成本将会成为很大的开销。

构建大数据存储引擎的必要性在于：

（1）数据不断地增长，成 TB 和 PB 级别。

（2）横向扩展相对比较容易，只需要增加相应的机器节点即可，可使用数据的不断增长需求。

（3）需要支持随机的读 / 写请求。

（4）传统关系型数据库相对比较昂贵，数据扩容相对比较复杂。

4.1 架构体系

HBase (Hadoop Database) 是一个开源的非关系型分布式数据存储系统，也是基于列存储模型的分布式数据库，是隶属于 Apache 基金会旗下的项目。HBase 具备高可靠性、高性能、面向列、可伸缩的特征，它是 Google BigTable 的开源实现，采用了 BigTable 的数据存储模型：增强的稀疏排序映射表 (Key-Value)，HBase 提供了对大规模数据进行随机、实时读 / 写访问。同大多数分布式系统一样，采用 Zookeeper 作为系统服务。

4.1.1 结构概要

大数据存储系统都是采用 NoSQL 的理念，区别于传统的关系数据库，相比关系型结构，NoSQL 具备高性能、易扩展、数据模型灵活等特征。关系型数据更注重数据关系，NoSQL 更注重数据存储。

(1) 高性能。由于无关系性，导致数据库的结构相对简单，具备很高的读 / 写性能，相比关系型数据库，NoSQL 可以部署在廉价低配的服务器，而关系型数据库常常需要特定性能配置的服务器。

(2) 易扩展。去掉了关系数据库中的关系，数据之间变得相对独立，也正是因为如此，扩展性大大增强，而关系型数据库需要考虑各种约束情况。

(3) 数据模型灵活。传统的关系型数据库需要预先定义字段，而 NoSQL 无须事先为存储的数据定义字段，更加灵活地控制字段，关系型数据在存储大量数据的情况下，更改字段是非常困难的事情。

HBase 可以通过增加服务器使得存储更多数据，满足搜索引擎的数据增长需求，HBase 还具备自动故障切换、高一致性、高性能随机读 / 写特性。HBase 中的表特点如下：



- (1) 表很大。一个表可以存在上百万列，也可以拥有上亿行的数据。
- (2) 数据是面向列的存储和权限控制。
- (3) 稀疏。对于允许存在的空列（null），并不占用存储空间，表的设计非常稀疏。
- (4) 表中的数据元素都是字符串类型。
- (5) 表中的每个单元格可以按照单元格插入的时间戳存储多个版本的数据。

随着大数据越来越能够体现数据价值，搜索引擎需要通过大数据发掘最具价值信息给用户，而不是大量结果，大数据的存储，是搜索引擎完成此项工作的基石，优秀的 HBase 通过良好的性能和分布式实时计算作为一块基石，保障了搜索引擎对于海量数据的处理能力。

HBase 的基础体系架构如图 4-1 所示。

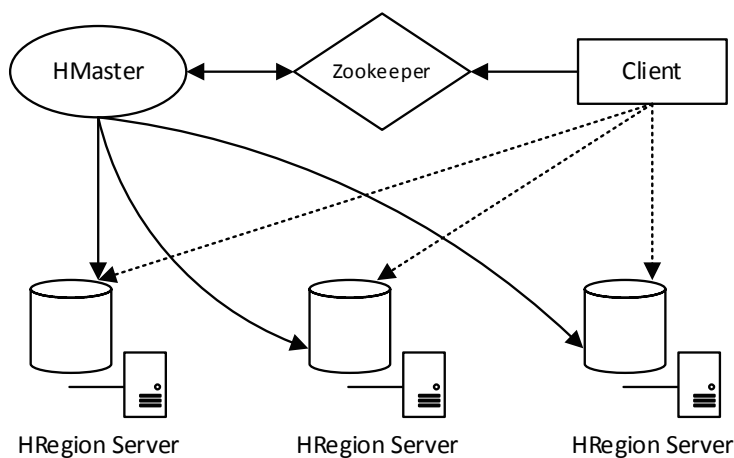


图 4-1 HBase 的基础体系架构

由图 4-1 可知，HBase 的基础体系架构包含 HMaster、Zookeeper、Client、HRegion Server 等相关组件。

(1) Client。HBase 的数据访问接口，Cache 的维护者，并利用 Cache 加快对 HBase 的数据访问。

(2) Region 表示数据区域，每个表一开始仅有一个 Region，但是随着数据量的不断增加，当达到一定阈值之后，此 Region 将会被切分为两个新的 Region，换个角度理解即是：一个表的数据量非常大，不可能存放在一台服务器上，会根据数据大小进行切分，保存为多个 Region，并存储在不同的服务器上（Region Server）。

(3) Zookeeper。是一款基于分布式的开源应用程序协调服务，在分布式计算中，主要用它实现同步服务、配置维护等，Zookeeper 部署的机器节点越多，则可靠性越高。在 HBase 中用于协同服务的工具，存储所有的 Region 的寻址入口并实时监控 Region Server 的上线和下线信息，并将信息转发给 Master，以及存储 HBase 的 Schema 和 Table 元数据信息。利用 Zookeeper 保证了在 HBase 环境中只存在一个 HMaster。

(4) HMaster。HBase 中 HMaster 可以存在多个，通过 Zookeeper 的 Master Election 机制保证。HMaster 主要对 Table 和 Region 进行管理。包括为 Region Server 分配 Region，调整 Region 的分布情况，保障 Region Server 的负载均衡。当有 Region Server 不能正常工作时，将该机器上的 Region 进行迁移。以及管理用户对 Table 的增加、删除、修改、查找等操作。

(5) HRegion Server。是数据的实际存储节点，负责维护 Region，处理对这些数据的各种请求操作，也负责切分在运行过程中变得过大的数据文件。

HBase 中很重要的一个思想是利用了 Google BigTable，BigTable 为搜索引擎提供爬虫抓取的网页提供存储服务，BigTable 提供按行级别的访问，所以爬虫不仅可以插入网页文档，还可以更新某个网页文档。这也是一般开源搜索引擎采用 HBase 的原因。



4.1.2 服务器上线

在整个 HBase 体系启动过程中，存在服务器上线流程，主要针对 HMaster 及 HRegion Server。

(1) HMaster 上线流程。HMaster 启动之后，会在 Zookeeper 上获得一个标识 HMaster 的锁，表明自己已经是 HMaster，然后获取 Zookeeper 中记录的所有数据服务器列表，并和每个数据服务器进行通信，将数据资源区域开始分配，并获得数据服务器与数据区域的对应关系。

(2) HRegion Server 上线流程。HMaster 会根据 Zookeeper 进行了解各个数据服务器的状态，新的数据服务器启动之后，会在 Zookeeper 的 Server 目录下面建立属于自己的文件，并享有该文件的独占锁，HMaster 监听了 Zookeeper 的 Server 目录的消息信息，即当 Server 目录下有任何增删改的操作，HMaster 都可以收到消息，并且是几乎实时的消息，如若收到的是 Server 下创建的新的目录，则 HMaster 收到了一条新数据服务器上线的消息，后面 HMaster 将会开始进行安排数据让其存储。

4.1.3 服务器下线

在 HBase 运行过程中，也存在 HMaster 及 HRegion Server 下线流程。

(1) HMaster 下线流程。HMaster 维护了表信息和数据文件与数据服务器的对应关系，因此 HMaster 下线，意味着无法进行表操作，也不能对数据区域进行负载均衡，还包括数据文件的合并等，一切依赖 HMaster 直接执行或指挥执行的任务均将终止，但是针对数据的读 / 写依然可以正常进行，因为数据文件服务器依然正常工作。

(2) HRegion Server 下线流程。当数据服务器下线时，它会和 Zookeeper

断开连接，无论是正常自动断开还是异常断开，Zookeeper 收到断开消息，将会自动释放这台数据服务器占有的 Server 目录下属于该数据服务器的文件独占锁。而 HMaster 也在不断扫描各个数据服务器的文件锁状态，如果发现某台数据服务器的独占锁已经释放，HMaster 则会尝试去获取该数据文件服务器标识文件的读 / 写锁，一旦获得，则确定该数据服务器发生退出或异常情况，此刻 HMaster 会将数据服务器在 Server 下的标识删除，表示这台数据服务器已经不能提供数据服务，并将退出的数据服务器的数据（Region）转给其他数据服务进行服务。

4.1.4 数据读取

对于 HBase 的数据写入过程如下：

首先是把 Log 写入到 HLog 中，HLog 是标准的 Hadoop Sequence File，由于 Log 数据量小，而且是顺序写，速度非常快；同时把数据写入到内存 MemStore 中，成功后返回给 Client，所以对 Client 来说，HBase 写的速度非常快，因为数据只要写入到内存中，即算成功。

接着检查 MemStore 是否已满，如果满了，就把内存中的 MemStore Flush 到磁盘上，形成一个新的 StoreFile。当 StoreFile 文件的数量增长到一定阈值后，系统会进行合并（Compact），在合并过程中会进行版本合并和删除工作，形成更大的 StoreFile。当 StoreFile 大小超过一定阈值后，会把当前的 Region 分割为两个（Split），并由 HMaster 分配到相应的 HRegion Server，实现负载均衡。

对于 HBase 的数据读取过程：

由于无法直接修改 HBase 中的数据，所有的 update 和 delete 操作都转换成 append 操作，而且 HBase 中也没有索引，因此读数据的方式均采用 Scan 方式进行。Client 在读数据时，一般会指定 timestamp 和 ColumnFamily。首先，根

据 ColumnFamily 可以过滤掉很大一部分 Store，这也是 HBase 作为列式数据库的一大优势。然后，根据 timestamp 和 Bloom Filter 排除掉一些 StoreFiles，最后，在剩下的 StoreFile（包含 MemStore）中采用 Scan 方式查找。

4.2 数据模型

数据存储模型，HBase 会自动将表划分为不同的几个区域，每个区域即为不同行数据集，按照存储的主键进行划分。一个数据区域（HRegion）的区分标识由表名、开始主键、唯一 ID 组成。Region 中的数据是连续的数据，从表的角度来看，表内数据顺序并没有改变，仅仅是作了面向行的区域划分，这些被划分的数据存储在数据服务器（HRegion Server）上，如图 4-2 所示。

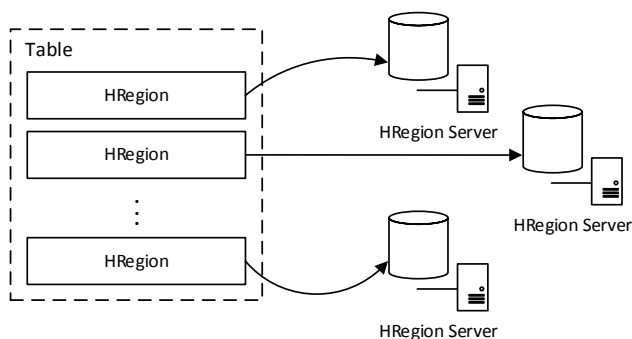


图 4-2 数据存储模型

从数据存储服务器上，一台服务器是一个 HRegion Server 的实例。每个数据区域仅存放在一个数据服务器中。从逻辑视图中看，数据区域是表的最小存储结构，但是从数据服务器上的物理存储来看，它由更多的存储单元组成（HStore），针对数据区域的物理存储结构如图 4-3 所示。

数据服务器上存储了大量不同的数据区域，每个具体的存储单元由两部分构成：MemStore 和 HFile，用户新插入或者更新的数据，会先存储到

MemStore，当 MemStore 达到一定值时，会被写到存储文件（StoreFile），存储文件底层采用 HFile 实现。

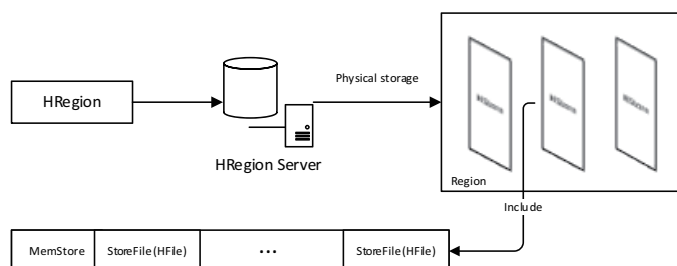


图 4-3 针对数据区域的物理存储结构

但是存储文件也不是无限制的数量增长，当文件大小增长到一定的阈值，多个存储文件会合并成一个存储文件，值得注意的是，数据的合并并不是简单的合并，是根据数据的版本合并及数据删除，HBase 的数据在执行删除操作的时候，实质当时并没有进行数据删除，而是在合并的时候才真正删除，用户的操作只会到达 MemStore 一层，因此与用户的交互性能非常高，随着数据文件的合并，存储文件也会越来越大，存储文件的大小也会触发数据文件的拆分，当文件大小达到一定值之后，数据服务器上存储的数据文件会被分为两个新的数据文件，被拆分的数据文件下线，两个新的数据文件上线，缓解数据压力，如图 4-4 所示。

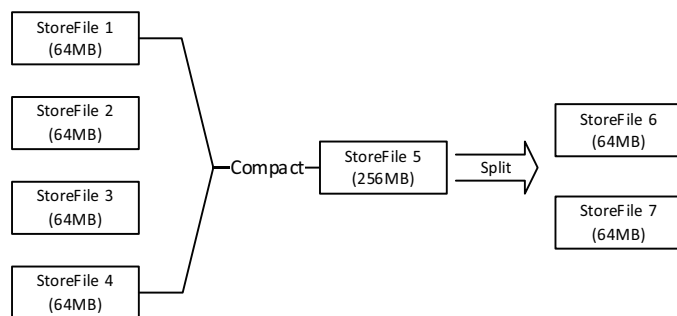


图 4-4 文件合并拆分示例

在被存储的数据中，除针对用户存储的数据之外，HBase 还会通过日志记录文件，记录用户的数据操作行为。



HLog 是 HBase 的日志记录文件，对于分布式存储，日志一般用于容灾处理、数据恢复，HLog 用于记录所有数据的变更，每个数据服务器都维护着一个 HLog，之所以不是针对每一个表进行日志记录，采用数据服务器级别的记录，是因为在庞大的数据操作中，仅仅对一个日志文件操作，而不是对多个日志文件操作，可以减少磁盘的寻址次数，也许几次操作不明显，但是海量的数据操作就会显示出针对数据服务器级别的日志记录优势，提高对数据操作的性能。

一旦数据服务器发生异常退出，根据上面的存储结构，用户最新的操作在内存中，则这部分内存即刻被释放，将会造成数据丢失，在任何一个分布式存储中，数据丢失都是不允许的。鉴于每个数据服务器都存在一个 HLog 对象，当每次用户进行操作的时候，会先在内存操作（MemStore），在操作的同时，也会写一份文件到 HLog 文件，而 HLog 文件也会不断更新，例如，已经被写到磁盘上的存储文件中的数据将会被清理，当数据文件服务器发生意外之后，HMaster 可以监听到，并且确认数据服务器是否真的出现异常，HMaster 将会首先处理 HLog 文件，将不同的 Region 的数据进行拆分，分别放置到 Region 目录下，最后将这些丢失的 Region 进行重新分配，被分配到的数据服务器，发现有属于自己的 HLog 需要处理，因此会将对应的数据加载到自己的 MemStore 中，完成数据的容灾恢复。

4.3 数据压缩

在 HBase 中存储的文件支持三种压缩算法，分别是 GZIP 压缩算法、LZO 压缩算法及 Snappy 压缩算法，三种算法中，GZIP 在压缩率上相对优于 LZO、Snappy，但是对 CPU 资源的消耗也相对较高，然而对于相同数据分别在 GZIP、LZO 解压缩速度下 Snappy 性能是最优的，LZO 的中和性能处于 GZIP

和 Snappy 之间。在不同的应用场景中，三者都有各自的优势。

(1) GZIP 压缩算法。优点在于拥有较高的压缩率，处理 gzip 文件和直接读取文本文件处理一样，应用广泛，大多数 Linux 系统都带有 gzip 命令。缺点是不支持压缩后数据分割。

在应用过程中，最好压缩的数据文件大小在系统的处理范围之内，适合采用 gzip，在 HTTP 请求中，就有对网页等相关数据进行 GZIP 压缩的要求，然后再传输，降低了网络传输数据量，从而提高了浏览器客户端的访问速度。对于分布式存储，则需要确定压缩后的数据文件块大小必须在允许的块大小范围内，强行分割将会出现数据异常。

(2) LZO 压缩算法。优点在于解压缩速度较快，支持文件分割，是分布式存储中比较流行的压缩方式。缺点在于压缩率相对 GZIP 较低，由于 LZO 使用的是 GPL 开源协议，而 Hadoop 采用的是 Apache 开源协议，导致协议之间有条件冲突，因此 LZO 需要另行安装。对于单个较大的文件，使用 LZO 比较合适，文件越大，压缩效果越明显。

(3) Snappy 压缩算法。优点在于很高的解压缩性能，较为合理的压缩速率，不会占用大量 CPU。缺点在于不支持压缩后对文件进行分割，压缩率比 GZIP 低，Linux 系统下没有相应命令。为了更高的解压缩速度，它允许牺牲少量压缩率，以达到综合性能最优，适合于大数据处理中的临时中间文件或者数据，中间数据的处理应该是越快越好，而这正是其解压缩性能能够带来的。

4.4 负载均衡

不同于分布式计算的负载均衡，分布式存储时，数据存储的地方不会是个，因此，负载均衡协调的是单个服务器在集群中的任务量，而对于分布式实

时计算，每个任务都会在三台服务器间协同指向，它的负载均衡是保障多个服务器在集群中的任务量分布。

对于任何一个分布式存储系统来说，负载均衡都是非常重要的部分，有效地保证分布式环境处于负载均衡的状态，可以有效地利用网络带宽、磁盘 I/O 及数据处理能力，以达到最佳化利用系统资源、最大化吞吐率、最短响应时间，同时避免过载导致系统崩溃等。

HBase 存在默认的负载均衡算法，以每个数据服务器的管理的数据区域作为任务量，通过将数据区域的数量均衡的负载给每个数据服务器作为负载均衡的条件。例如，存在三台服务器，它们目前已经管理存储的数据区域如图 4-5 所示。

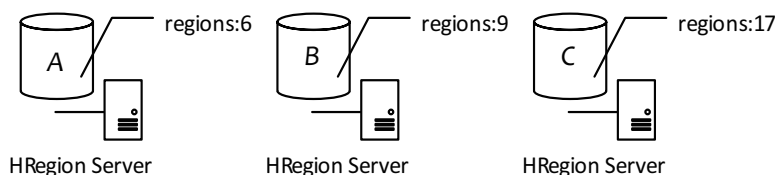


图 4-5 数据存储服务器中已经存储的任务量分布示例

三台数据服务器 A、B、C 分别对应的任务量为 6、9、17，直观情况存在过载问题。负载均衡的条件是三台服务器上的任务量基本一致，三台服务器累计任务量是 32，平均应该拥有的任务量约为 10.7，即每台任务量在 10 或者 11 上比较合理，设定每台服务器均衡负载的任务量中： $\min=10$ ， $\max=11$ 。

将数据服务器以已有任务量进行从小到大排序，依次为 A、B、C。从拥有任务量最大的数据服务器开始遍历，用它的任务量减去负载均衡后的最大允许数量，即 $17-11=6$ ，则意味着，它需要拿出 6 个任务，它的任务量变为 11，如图 4-6 所示。

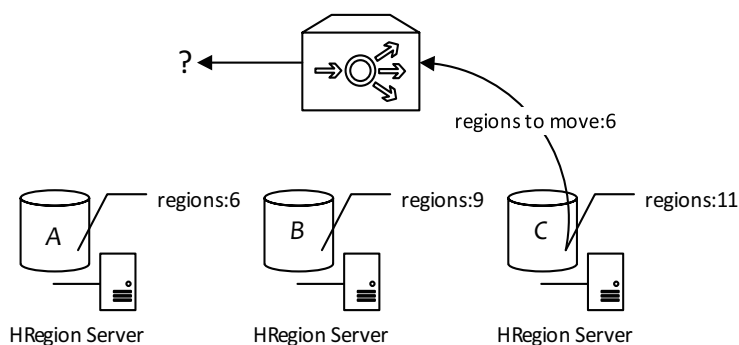


图 4-6 对任务量最大的数据服务器进行负载均衡

因此记录器中记录了，需要从 *C* 中拿出 6 个任务，但是任务还未重新分配出去，再次从任务量最小的数据服务器开始遍历，当找到一个负载量比最小值大时停止，在这个过程中，将还未分配的负载量，依次分配给数据服务器（*A*、*B*），使得分配后的数据服务器负载量和最小值相等（达到 10），如图 4-7 所示。

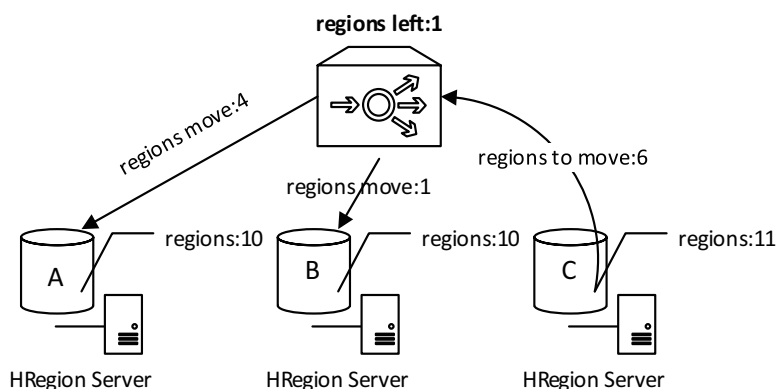


图 4-7 对任务量较小的数据服务器进行负载均衡

经过调整后，数据服务器 *A*、*B*、*C* 目前负载量分别为 10、10、11，但是依然有 1 个负载任务并没有分配出去，此刻需要检查当前负载均衡的工作，确保没有数据服务器的负载量小于设定的最小值，再次从最小负载量的数据服务器（重排序后）开始遍历，直到数据服务器的负载量刚好等于约定的负载均衡的最大值（11）停止，在本次遍历的过程中，凡是负载量没有达到最大值的数



据服务器，均再次分配负载任务，直到和最大值相等，以及没有负载任务可以分配为止。最终负载均衡任务量分布如图 4-8 所示。

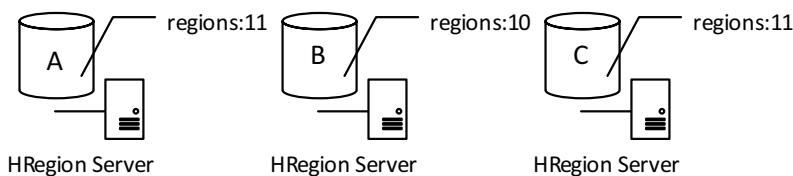


图 4-8 最终负载均衡任务量分布

上述为 HBase 中的默认负载均衡方式，也是分布式系统中常用的分配方式，在深入扩展的情况下，可以将服务器配置、数据访问量也作为其中参考因素，进行综合的均衡负载。

4.5 数据存储逻辑视图

在进行存储示例介绍之前，首先需要了解存储的数据内部模型，对于一条数据的存储，结构如图 4-9 所示。

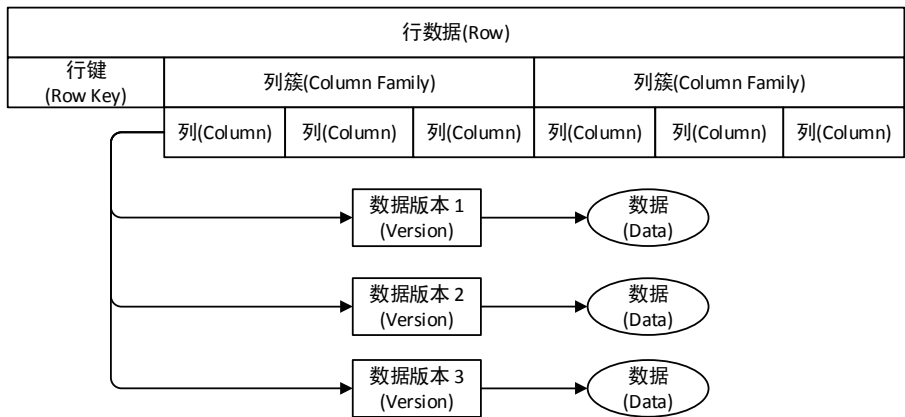


图 4-9 HBase 中数据存储的结构示例

图 4-9 中，分别包含行键、列簇等相关含义，分别如下所示：

(1) 行键。每一行数据都有唯一的行键，行键没有对应的数据类型，在内部可以被视为一字节数组，用来检索记录的主键，它可以是任意字符串，允许的最大长度为 64KB。

(2) 列簇。数据在行中被组织成列簇，列簇是表结构的一部分，在使用之前必须定义列名都是以列簇作为修饰前缀，例如列“car:suv”“car : mpv”都属于“car”这个列簇。

(3) 数据版本。每列都有属于自己的数据版本信息，版本数据可以配置。版本则通过时间戳进行索引，时间戳的类型是一个 64 位整型，在存储数据时由系统自动或用户显示赋值，时间戳的精确度为时间的毫秒级别。为避免数据存储过程中，过多的数据版本导致数据管理加重，因此 HBase 提供了两种数据版本回收机制，一方面通过保存数据的最后 N 个版本，另一方面保存最近一段时间内的版本（如最近一天或者最近一周等）。

根据图 4-9 所示的存储逻辑结构，HBase 中针对逻辑结构的存储网页部分数据示例如表 4-1 所示。

表 4-1 HBase 中针对逻辑结构的存储网页部分数据示例

行键	列簇（“Document”）			列簇（“Description”）	
	列 （“Title”）	列 （“Content”）	列 （“Author”）	列簇 （“MimeType”）	列簇 （“...”）
a.com	T1：“大学排名…”	T1：“大学排名对社会…”	T1：“李明”	T1：“TEXT/HTML”	...
	T2：“大学排名…”	T2：“大学排名争议…”	T2：“李明”	T2：“TEXT/HTML”	...
b.com	T1：“手机辐射…”	T1：“手机辐射分析…”	T1：“张杰”	T3：“TEXT/HTML”	...
	T2：“手机辐射…”	T2：“手机辐射结论…”	T2：“张杰”	T4：“TEXT/HTML”	...
	T3：“手机辐射…”	T3：“手机辐射建议…”	T3：“张杰”	T5：“TEXT/HTML”	...

如果将上述存储信息按照关系型数据库的逻辑存储结构进行存储，则存储示例如表 4-2 所示。

表 4-2 利用关系型数据库存储 HBase 中同样的数据示例

主键 (URL)	列：时间戳 (Timestamp)	列：标题 (Title)	列：内容 (“Content”)	列：作者 (“Author”)
a.com	T1	“大学排名…”	“大学排名对社会…”	“李明”
a.com	T2	“大学排名…”	“大学排名争议…”	“李明”
b.com	T1	“手机辐射…”	“手机辐射分析…”	“张杰”
b.com	T2	“手机辐射…”	“手机辐射结论…”	“张杰”
b.com	T3	“手机辐射…”	“手机辐射建议…”	“张杰”

观察表 4-1 和表 4-2，可以发现两者对于数据存储的关系存在不同的方式，除此之外，对于 HBase 中大数据面向列存储及关系型存储的区别还包括如下：

(1) 关系型数据库中每列必须定义数据类型（如整型、可变长字符串等），而在 HBase 的类 BigTable 的存储中，有且仅有一种数据类型即字符串类型。

(2) HBase 中不存在表与表之间的关系，而在关系型数据库中，可以建立表与表之间的关系，并且还可以通过 SQL 语句进行跨表数据操作（如查询、删除等）。

(3) 由于 HBase 对于列的划分不仅仅是普通的列，在列之上还包括列簇，每个列簇都有多个文件保存，不同列簇的文件也是分离的，这与关系型数据库中的表格结构和行存储不同。

(4) HBase 中存储的数据都带有时间戳，因此，HBase 对数据的修改和删除实质上并没有对原始数据进行修改或者删除，只是新增一条时间戳日志的记录。而关系型数据库则是对数据内容的修改和删除。虽然关系型数据库本身不带有时间戳数据的概念，但也可以通过自定义时间戳完成，不足的是需要自动维护基于时间戳的历史数据并且数据带有较大的冗余。

(5) HBase 摒弃了关系型数据库中的一些复杂操作，使得达到高性能的目

的。在一般情况下，HBase 的数据操作性能优于关系型数据库。

上述中行键是 HBase 中比较重要的结构，存储过程中数据按照行键的字典序排序存储，因此在设计行键的时候，要充分利用该排序特性，将可能经常一起读的数据存放在一起。可以将行键视为关系型数据库中的主键，在对 HBase 中的表数据进行访问时，存在三种方式：

- (1) 通过定位某个行键进行访问行数据。
- (2) 通过行键的范围访问某些数据。
- (3) 通过全表的扫描获得相关数据。

关系型数据库和 HBase 都有相互独特之处，但是对于搜索引擎的数据进行存储，若采用关系型数据库，不仅影响了存取性能，还不能完全满足数据增长的动态扩容。因此，HBase 对于搜索引擎而言，可以快速地进行海量数据存储，并在性能上达到较好的效果。

4.6 本章小结

海量数据存储的价值不仅仅是将数据存储，而且还能够进行高性能的操作，没有海量数据的搜索引擎不叫搜索引擎，仅仅只能称为检索系统，检索系统的数据存储要求并不高，通过关系化的结构数据库几乎就能解决其问题，海量数据给予搜索引擎更高的价值。HBase 作为一个优秀的海量存储引擎，非常适合用于搜索引擎的网页原始数据存储。本章针对 HBase 在大数据分布式存储的体现结构和关键信息进行了详尽描述，更多关于 HBase 细节可参考 Apache HBase 的官方网站。

第 5 章 构建分布式实时计算

在搜索引擎中，后台系统在不间断地进行数据抓取、索引建立等工作，搜索引擎一直在尽可能让新抓取的页面能够被用户在最短时间内搜索到，也试图让被删除的网页尽可能从搜索引擎中尽快移除出去。在当前移动互联网盛行，实时计算变得更加迫切，用户需要尽可能通过搜索引擎发现更新的新闻、更实时的天气预报、更准确实时的交通信息等，通过分布式实时计算，可以减少这样的事件延迟。

5.1 概述

分布式实时计算一般都是针对海量数据进行的复杂运算，包括实时输入和实时输出。目前包括分布式实时计算在内的大数据处理解决方案如表 5-1 所示。

表 5-1 目前包括分布式实时计算在内的大数据处理解决方案

解决方案	组 织	类 型	概 要
S4	Yahoo !	流式计算	一个通用的解决搜索广告展示及用户点击反馈的流式系统
Storm	Twitter, Apache	流式计算	是一个免费、开源的分布式实时计算系统，用于实时日志处理
HPCC	LexisNexis	批量计算	一个大规模并行处理计算平台，用于解决大数据问题
Hadoop	Apache	批量计算	实现在大量计算机组成的集群中对海量数据进行分布式计算

续表

解决方案	组 织	类 型	概 要
Spark Streaming	Apache	流式计算	属于 Spark 的核心 API，支持高吞吐量、支持容错的实时流数据处理

在当前时代越是在海量数据面前，各大互联网公司越是希望能够尽快发现数据价值，并将价值展现给用户，例如，在电子商务、新闻推荐、社交媒体、数据监控、日志分析等领域应用广泛。尤其是在当下大数据时代，数据挖掘和机器学习都根深蒂固地在每个互联网公司，成为实时为线上服务提供数据支持的关键，而这些都需要分布式实时计算作为铺垫。

搜索引擎时时刻刻都在进行着计算，构建一个分布式实时计算系统，必须要关注如下五个问题：

(1) 具备高吞吐率。能够在单位时间内处理尽可能多的数据，包括数据读取、数据分析、数据最终的存储整个流程。

(2) 具备分布式。能够协同多机工作，在各个机器之间任务相互独立。在集群的环境中协同工作是分布式系统最基本的要求。

(3) 低延迟性。既然是实时系统，那么低延迟是理所当然。需要其能够在尽可能短的时间内完成海量数据处理，并将处理结果及时反馈到相应请求系统。

(4) 可扩展性。随着业务的发展，数据的计算复杂度可能会增大，计算的服务器会相应进行扩容，在新增服务器的情况下，保证当前已经允许的系统持续有效工作，不影响系统中的任务正常运行。

(5) 容错性。服务器宕机等不可预知的风险都会存在，但是不得因为服务器中断而导致任务终止执行，系统需要具备能够有效将失败数据进行重新计算或者估值计算的能力。

过去的十年是数据变化最快的十年，最初许多公司都采用 Hadoop 及一些



其他关键技术处理大数据，但这些技术都不是实时系统，而且它们的设计初衷也不是为了实时计算。因此已经存在一些分布式实时计算引擎，例如，Apache Storm、Storm 的出现填补了大数据处理在实时计算中的缺失。通过自行设计符合搜索引擎系统的分布式计算引擎，可以助力完善搜索引擎的不足。本章以 Storm 为案例，讲述分布式实时计算的实现及在搜索引擎中的使用。

Storm 是完全开源的分布式实时计算框架，采用 Clojure 语言编写。Clojure 是一种基于 Java 平台的动态函数式编程语言，其语法与人工智能领域的 Lisp 相近，在 Java 平台运行时，程序也被编译为基于 JVM 的字节码进行运算。Storm 属于当前互联网行业中最流行的分布式计算框架之一，起初是由 BackType 公司研发的实时处理系统，后来 BackType 公司被 Twitter 收购，现在 Storm 是 Apache 基金会的项目成员之一。

5.2 设计架构

分布式实时计算能够进行实时计算的基础在于良好的设计思想和体系结构，本节以 Storm 的设计思想与基本架构为基础做相应介绍。Storm 是基于数据流的分布式实时处理系统，拥有强大的数据吞吐能力和实时计算能力，它提供了完整的数据处理机制、容错机制，不仅具备高性能、高容错性、可扩展性，还支持多语言性。可以满足大数据实时数据分析的业务需求。

5.2.1 设计思想

分布式实时计算中，数据在内存中被连续计算，形成数据流，数据流被经过的各个处理池处理之后，流向新的处理池或者达到处理效果直接输出结果。数据流的起始端（Spout）如水龙头，不间断地供给处理的流（Stream），数据

流经过各个处理站（bolt）依次被处理，如图 5-1 所示。



图 5-1 流式计算的设计原型示意

每个数据流处理池能够处理来自各个地方的数据流。Storm 根据这样的思想，抽象出计算模型，如图 5-2 所示。

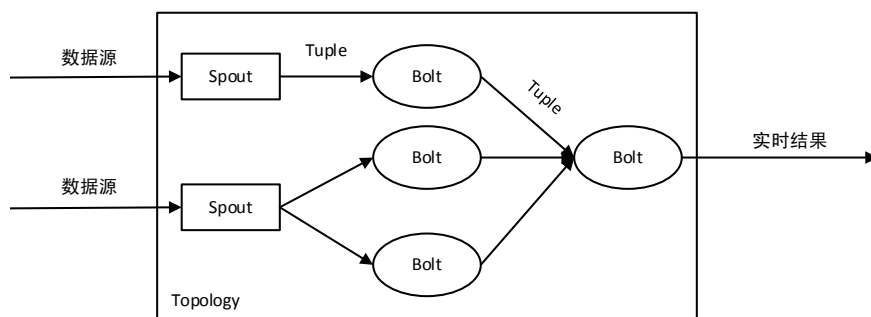


图 5-2 流式计算的计算模型示意

在 Storm 计算模型中，一个实时应用程序称为 Topology，原因是其运行实质是一个拓扑结构。拓扑结构中有两个非常重要的消息流动组件：Spout 和 Bolt。

（1）Spout。是数据的产生源，一般情况下读取外部数据（例如日志信息）后通过 Spout 处理成 Topology 的数据源。Spout 根据数据源的情况，可以源源不断地产生数据、处理数据，这个源源不断的数据由一个个 Tuple 组成，Tuple 是组件之间的数据传送单元，Tuple 将会被指定的 Bolt 接收并处理。

（2）Bolt。是主要负责处理数据的组件，可以进行数据求和、合并、写缓存等任意操纵，Bolt 的数据源来自 Bolt 或者 Spout。Tuple 在组件之间相互传输过程中，一个 Bolt 可以接收多个 Tuple 源。之所以将图 5-2 称为流计算，也是因为大量的 Tuple 在拓扑结构中流动，形成了数据流，数据流在各个节点之间分别处理，类似于流水作业。

5.2.2 基本框架

著名的开源分布式实时计算系统 Storm 的框架如图 5-3 所示。

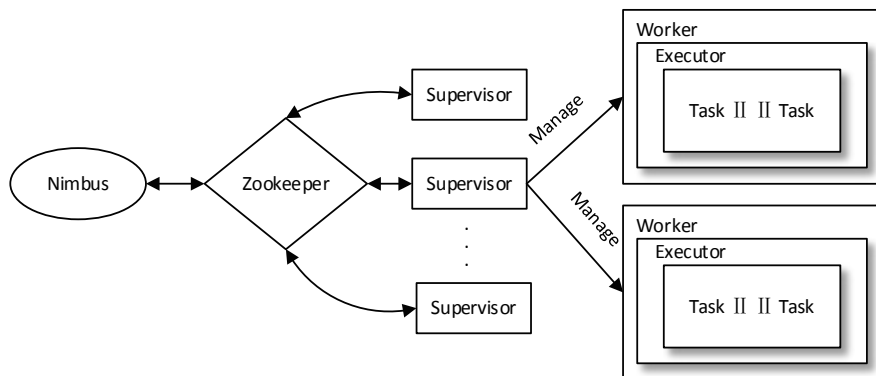


图 5-3 开源分布式实时计算系统 Storm 的框架

根据图 5-3 所示，在 Storm 的整个计算框架中，存在 Nimbus、Supervisor、Zookeeper 等相关概念。

(1) Nimbus。用于负责整个分布式系统中的资源分配和任务调度。主要包含三方面任务。第一，它是整个 Topology 开始的地方，将 Topology 的 jar 提交到整个集群中运行；第二，当进行提交 Topology 的时候，进行资源分配、任务调度；第三，当有 Topology 正在运行的时候，监听所有 Worker 的心跳情况，如果 Worker 存在崩溃的情况，则重新分配资源。

(2) Zookeeper。Storm 系统中依靠 Zookeeper 集群实现节点间的信息交换。Zookeeper 的路径节点同文件系统的路径节点相似，可以通过这个节点存储数据，也可以创建子节点，是一个目录式存储结构。

(3) Supervisor。负责接收 Nimbus 分配的具体任务安排，控制 Worker 进程的启动和停止。

(4) Worker。是工作任务的具体进程，是一个 Topology 的子集，一个 Worker

只能为一个 Topology 服务，它会启动一个或者多个 Executor 以线程为单位执行 Spout 或者 Bolt。

(5) Executor。是在 Worker 中的执行线程，每一个 Executor 只会执行一个 Spout 或者 Bolt。

(6) Task。是任务中的每一个具体任务，一个 Task 是 Spout 或者 Bolt 的实例，Executor 会执行 Task 里面的函数方法，例如 nextTuple、execute。

上述中，Nimbus 和 Supervisor 都是 Storm 中提供的后台守护进程，可以在同一服务器中允许，Zookeeper 也可以与 Nimbus、Supervisor 安装在同一服务器中，但一般情况下 Zookeeper 则是在多个节点机器中协同允许，通过各个组件的协同工作，完成对源源不断数据的处理。但是分布式计算中都会有这样的问题，当数据流来了之后，会分发给两台甚至更多的服务，涉及为这些数据分组。数据流分组定义了一个数据流中的 Tuple 如何分发到 Topology 中不同 Bolt 的 Task 中，在 Storm 中，存在随机分组、字段分组、全部分组、全局分组、直接分组、无分组、本地或随机分组七种分组方法。

(1) 随机分组 (Shuffle Grouping)。随机分发 Tuple 到各个 Bolt 上执行任务，但是会使各个 Bolt 获得的 Tuple 数据相等。

(2) 字段分组 (Fields Grouping)。依据指定的字段进行切分数据流。保证具体处理的 Task 收到的是字段相同的 Tuple。主要应用于数据的去重、关联 (Join) 等操作。

(3) 全部分组 (All Grouping)。广播式发送，每个 Tuple 数据单元都会被发送到 Bolts 中。

(4) 全局分组 (Global Grouping)。整个 Stream 中的所有 Tuple 都会被汇集到 Bolt 的一个具体 Task 中，一般选择汇集到 ID 值最小的 Task 中。



(5) 直接分组 (Direct Grouping)。Tuple 的发送者必须指定接收 Bolt 的具体 Task 处理此消息，需要将 Stream 声明成 Direct Stream 才允许使用此分组方式。

(6) 无分组 (None Grouping)。效果和随机分组相似，为预留的分组方式。

(7) 本地或随机分组。与随机分组类似的分组方式，不同点在于 Storm 会把这个 Bolt 放到这个 Bolt 相应的订阅者同一个线程中去执行。

除上述七种分组方式之外，还可以通过自定义分组的方式进行分组，主要通过实现 Storm 中的 Custom Stream Grouping 接口从而达到自定义分组的目的。

5.3 运行模式

将基于分布式实时计算平台编写的应用程序提交到分布式实时计算平台中时，需要考虑如下两方面问题：

(1) 如何保证编写的程序可以正常运行且运行结果符合预期。

(2) 在不确定运行结果的情况下，如何调试所编写的程序。

之所以会考虑上述问题，是因为在一个已经搭建好的数百台分布式集群环境中，去调试一个程序不仅浪费资源，也显得不高效。因此，一般分布式实时计算平台都会设定两种模式，分别是本地模式和远程模式。

(1) 本地模式。以单机的方式运行，从 Storm 的角度分析则是放在本地计算机的单一 JVM 进程中运行。虽然是本地模式，但是在本地计算机中模拟的一个分布式集群环境，它拥有分布式集群环境中的绝大多数功能。这是基于与真实的环境非常接近的优势，在本地模式下可以集成开发环境，对编写的分布式程序进行参数调整、算法优化等，使得最终允许的分布式应用满足预期，因

此又称为开发模式。在 Storm 中，可以通过使用 LocalCluster 对象完成本地模式下的分布式应用程序编写。

(2) 远程模式。在程序发布之后的正式上线运行时采用远程模式运行。远程模式从 Storm 的角度分析则是在正式的分布式集群中将编写的任务放置到整个拓扑结构中运行，因此又称为生产模式。

5.4 负载均衡

在分布式实时计算中，负载均衡尤其重要，一旦任务分配不均，将直接导致某台服务器计算量增大，性能上出现严重问题。负载均衡的算法很多，包括轮询算法、最少连接算法、响应速度算法、持续性算法等。

(1) 轮询算法。提交每个任务之后，通过轮询的方式依次发送给每台服务器，不必在意每台服务器请求的任务量，需要服务器硬件和软件在每台服务器中的配置一致。

(2) 最少连接算法。通过服务器内部处理连接数量的信息，当有新服务连接请求时，根据服务器的连接数量选择具有最少连接数的服务器，将新请求服务分配给该服务器执行，这种算法适合长连接的请求服务。

(3) 响应速度算法。负载均衡协调的设备对内部服务器发出一个探测请求，计算出响应速度最快的服务器，并将新的请求分配给它。这种方法能够较好地反映服务器在某一时刻的最佳状态，但是不能保证响应服务器和请求客户端连接也具备最快的响应速度。

(4) 持续性算法。指定从某一个或某一些客户端发出的请求，都发向某一台服务器处理，可以通过客户端 IP、请求的报头、请求的 Cookie 等方法进行划分。



并不是每种负载均衡算法都适合分布式实时计算，原因在于在分布式任务提交的时候，看似只有一个任务，但是实质上会被分解为多个任务并在不同机器上运行，不同于单个服务请求。

以 Storm 为例，资源分配负载不仅仅是考虑分配 Spout、Bolt 数量问题，还需要考虑每个 Bolt 中执行任务的具体线程，涉及执行器（Executor）和任务（Task）的数量，执行器即为一个线程，任务是执行器需要执行的具体事情，假设有 10 个任务，但是设置有 5 个执行器，则表示每个线程执行 2 个具体任务。执行器以线程为单位在运行，依附于具体工作进程（Worker）。因此资源分配包括给节点分配及给计算进程分配。

（1）给节点分配。Nimbus 进行整个 Topology 的工作量及任务数量计算，任务数量为 Spout 和 Bolt 的并发度之和。例如，一个拓扑计算中，存在一个 Spout 和 Bolt，并发度分别为 3、9，则累计任务数量为 12。Nimbus 会将计算好的工作量分配给 Supervisor，以任务量为具体的分配单位。

（2）给计算进程分配。Supervisor 会根据 Nimbus 分配的任务量再一次分配给它的工作进程。工作进程按照任务量对当前任务量进行排序，依次轮询将执行器分配给工作进程，同一个 Spout 或 Bolt 所属的任务会尽可能分到不同机器的工作进程中。

通过任务分配的方式使集群处于负载均衡的状态是最好的方式。

5.5 通信设计

网络节点之间的消息传递一般会涉及消息通道的建立及消息的编解码问题。在分布式环境中除基本的通信方式之外，还会涉及接口的远程调用问题。

5.5.1 基本方式

通信主要包括进程内部和进程外部两方面通信。

(1) 工作任务进程内部的通信机制。同一工作任务进程内部，Worker 采用 LMAX Disruptor，它可以负责同一节点上线程间的通信。Disruptor 是一个高性能、低延迟且简单的消息组件，用于线程间共享数据。Disruptor 采用环形缓冲区（Ring Buffer）用作数据存储结构，Disruptor 之所以比较快主要包括三方面。第一，对于锁的使用，以 Lock-free 方式，采用 CAS（Compare And Swap/Set）。CAS 是一种乐观锁，采用 CPU 级别指令，越过操作系统级别，效率较高；第二，缓存基于数组，缓存的长度为 $2n$ ，可以通过位运算直接取地址 $i \& (length-1)$ ；第三，预分配缓存对象，覆盖数据，而不是清除，减少了垃圾回收启动频率。

(2) 不同工作任务进程的通信机制。不同工作任务进程（Worker）采用 Netty。Netty 是一个高性能、时间驱动的异步非阻塞 NIO（New Input/Output）框架，可以用于异步非阻塞的 TCP、UDP 应用程序。

5.5.2 分布式远程服务调用

在 Storm 的内部数据通信过程中，采用 Thrift 进行数据调用，Thrift 是一种可以跨语言的可扩展服务框架，它实际上通过中间语言来定义服务接口和数据类型，然后通过 Thrift 编译器生成远程服务调用客户端和服务端，远程服务调用是指在服务端和客户端都存在一个方法的接口，但接口的实现是在服务端，客户端通过相同方法接口，从服务端获得计算结果。

目前较为常用的远程服务调用方法很多，包括基于 SOAP 消息格式的 Web Service，基于 JSON 消息格式的 RESTful Service 等。在 SOAP 或者 JSON 格式的消息中，SOAP 实质上是一种 XML，XML 相对传输体积较大、传输效率低，



JSON 虽然体积较小，但是目前还不足够完善。Thrift 的消息传递采用基于二进制的格式，相对于使用 XML 和 JSON 体积更小，对于大数据处理、高并发协同处理有极大的优势。Thrift 还可跨语言进行远程服务调用，因此 Storm 采用 Thrift 之后，可以选择 C++、Java、C# 等其他语言编写应用。Thrift 的定义脚本中支持的数据类型包括基本类型（bool、byte、i16、i32、double、string）、结构体类型、容器类型（list、set、map）、异常类型及服务类型，一个简单的 Thrift 脚本如下所示：

```
service Calculator extends shared.SharedService {  
    i32 add(1:i32 num1, 2:i32 num2)  
}
```

上述定义的是一个计算器的 Thrift 脚本，里面包含一个求两个数之和的方法。针对该脚本，服务端负责该方法的具体实现，而客户端负载该接口的应用。在应用过程中，客户端通过调用该方法，并将参数传递进去，服务端获得参数之后，返回两数之和的结果给客户端。

在 Storm 的远程调用服务通信过程中，主要在如下三个应用场景中使用 Thrift：

- (1) 用户通过客户端向 Nimbus 提交 Topology 任务时。
- (2) Supervisor 通过 Nimbus 下载 Topology 任务时。
- (3) Storm UI 通过 Nimbus 获得 Topology 允许情况的统计信息时。

5.6 容灾恢复

任何一个分布式系统，都存在节点崩溃或者运行单元发生异常退出甚至宕机等异常情况，分布式系统中，可能会发生如下异常情况：

(1) 执行任务的具体进程崩溃。当执行任务的具体进程崩溃（如内存溢出、程序本身异常），则直接重启退出的任务进程。在 Storm 中，如果执行任务的 Worker 崩溃，则 Supervisor 会重新启动 Worker，Nimbus 也会监听该 Worker 的心跳，若 Nimbus 无法监听到心跳，则 Nimbus 会安排其他机器启动 Worker 进程，并将任务分配给新的 Worker 进程。

(2) 执行任务的集群节点崩溃。计算节点存在宕机的可能性，为保证计算的继续正常运行，需要转移该机器上的任务集，Storm 中该节点中的 Task 均会因为超时导致数据没有正常完成，Nimbus 监听到超时之后，会将这些 Task 分配到其他机器节点中运行。

(3) 主节点故障。在一般分布式系统中，只设计一个主节点，如果分布式集群中设计多个主节点，则代价非常庞大，因此主节点设计不在于一个或多个，而是在于主节点崩溃之后，现有任务继续运行。对于 Storm，Nimbus 服务器故障之后，Supervisor、Worker 会继续保持运行状态，Supervisor 也会给崩溃的 Worker 进行重启，唯一的不足是，无法重新分配新任务和资源。

5.7 数据容错原理

作为分布式实时计算，计算结果是实时输出，因此，要保障每个数据的输出都是有效输出，输出结果无误。要保证每个消息都被正常处理，而没有在网络传输或者处理异常中数据丢失，则需要通过验证消息机制保证数据均被处理。

Storm 在数据容错处理中具有较好的策略。Spout 作为数据的源头，因此数据的可靠性机制将会从 Spout 开始验证。Spout 需要记录其发射出去的所有 Tuple 信息，以便于当 Tuple 流经的 Bolt 处理失败之后，Spout 能够进行重新发射数据。它通过 ACK 机制验证数据是否被处理，Spout 在每次发射 Tuple 的时

候，都会产生一个消息编号，为 64 位数字，Topology 中存在一个除 Spout 和 Bolt 之外的系统级组件，称为 Acker。Acker 的任务是记录消息编号对应 Tuple 的处理路径，Bolt 在处理 Tuple 的过程中，如果被正常处理则进行 ACK，失败可以同 Fail 确认处理失败。另一方面倘若在规定的时间范围内，Tuple 没有被处理完毕，则告知消息的产生源 Spout，此消息处理失败，会重新处理，如果多次处理均显示失败，则终止 Topology。

判断消息是否被处理完的依据就是 Acker 的验证机制，如图 5-4 所示的 Topology，Spout 在发射的 Tuple 中，指定相应的消息编号，例如，发射给 Bolt A 的消息编号为 1001（实际为 64 位数字），发射给 Bolt B 的消息编号为 1100，在 Bolt A 和 Bolt B 向 Bolt C 发射新的 Tuple 的时候，也产生了新的消息编号，分别为 1011 和 0110，最终 Bolt C 接收到相应数据，按照 Topology 的设计，消息处理到 Bolt C 没有产生新的 Tuple，即可认为消息处理完毕，如图 5-4 所示。

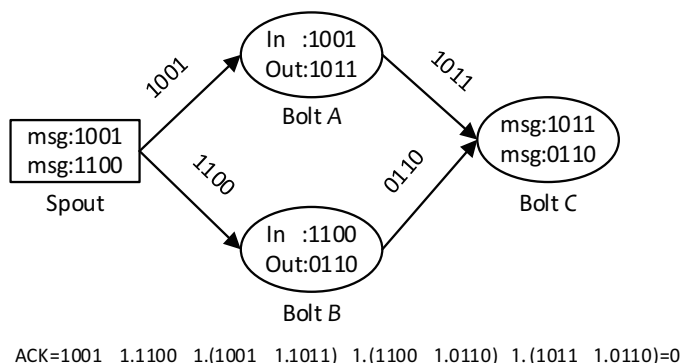


图 5-4 Storm 的 ACK 验证机制

而就是在整个数据处理过程中，Acker 记录下了处理路径，并将这些消息进行异或处理，先是每个 Spout 或者 Bolt 内部对传入的消息和传出的消息进行异或，然后将异或的值按照路径再次异或，若得到 ACK 的值为 0，则认为消息处理完毕，否则认为出现异常，利用数学的异或原理有效地将数据的处理或未处理进行了验证，实际上，消息的具体编号与最终的 ACK 是否为 0 没

有直接关系，因为消息编号在 ACK 的异或过程中均出现两次，相同值的异或为 0。

5.8 数据处理设计示例

分布式实时计算平台搭建完毕之后，开发人员会利用其实现各类数据分析和计算，根据前面介绍的思想，核心点是设计数据源的产生及数据的处理过程。依然以 Storm 为例，例如，对搜索引擎中当天的搜索词进行热点统计。热点统计的实质是词频统计，详细需求为：每天用户都在搜索引擎中输入大量搜索词，这些搜索词以句子或短语的形式存在，现在需要实时统计当天用户搜索的热点词汇，并按照搜索次数从高到低进行排序，忽略搜索词之间的语义相似度。

思路流程：首先通过一个 Spout 从搜索引擎的当天日志系统中获得用户最新的搜索词；然后以文本的方式发送给负责数据切分和清洗的 Bolt，处理完毕之后，负责数据切分和清洗的 Bolt 将处理之后的数据发送给负责词频统计的 Bolt，最终进行词频统计的 Bolt 将数据发送到一个负责汇总排序的 Bolt，最后负责汇总的 Bolt 将结果实时存储或实时上报，如图 5-5 所示。

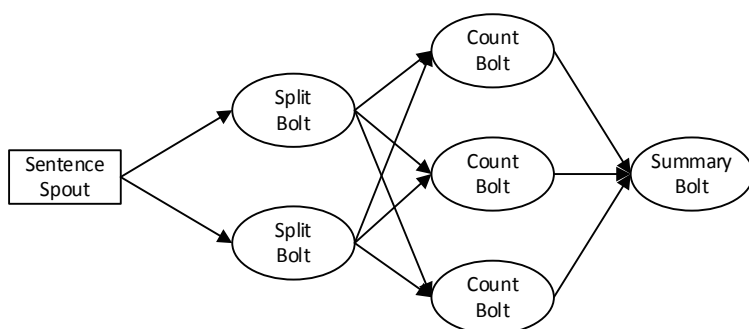


图 5-5 搜索引擎中热点搜索词统计分析的流式计算示例



分别实现 SentenceSpout、SplitBolt、CountBolt、SummaryBolt 即可，具体每个过程中处理的 Bolt 数量可以根据情况设定数量，在 Topology 中设置。

5.9 本章小结

以 Storm 为案例，概要介绍了分布式实时计算架构和设计理念，对于搜索引擎，处处都在进行计算，而几乎所有的地方，都希望能够实时产生结果，试图通过实时计算为用户提供最佳搜索体验。除此之外，人工智能机器学习也无时无刻地不在分布式计算中解决复杂问题。

利用分布式实时计算，在最短的时间内给用户提供最佳服务，告别了传统的方式，只注重分析处理性能，而不注重数据的查询性能，对于搜索引擎来说，实时反馈结果是必要条件之一，在海量数据面前，倘若没有分布式实时计算，采用传统的离线计算，导致的问题可想而知，尤其在广告推荐方面，将会给企业带来利益流失。实时计算不仅在解决搜索引擎技术问题，也在解决用户体验问题，良好的搜索体验是优秀的搜索引擎重要的体现之一。

本章不对 Storm 每个细节进行讲述，仅简述了分布式计算中可能涉及的关键问题，若想了解 Storm 更加详细的信息，可以参阅 Apache Storm 官网。

第 6 章 分布式可扩展爬虫

爬虫（Spider）作为搜索引擎的数据采集引擎，它通过开发者预先分配的种子 URL 集合，按照一定的规则自动化采集互联网资源，本质是一个网络数据请求下载工具。但是随着互联网的发展，互联网的数据显示出指数增长，现代搜索引擎对于爬虫的要求越来越高，能够抓取互联网资源仅仅是最基本的需求，还需要能够在尽可能短的时间内完成资源抓取任务，不仅如此，抓取效果也是非常重要的指标。

6.1 爬虫体系架构

在爬虫设计过程中需要利用分布式计算将网络爬虫的任务进行分解，利用集群中的服务器在分布式环境中快速获取资源。在大数据互联网时代，若爬虫仅仅通过单服务器，即使在单服务器中采用并行计算或者多线程技术，要实现大规模数据抓取，基本上是不可能的事情。分布式爬虫是技术发展的必然结果，依靠分布式计算平台，不仅能够高效持久地进行数据抓取，还可以动态地扩展机器，保证容灾。一般在实现架构中，有主从分布式结构爬虫和对等分布式结构爬虫，二者各有优势，也各有缺陷。

6.1.1 主从分布式结构爬虫

主从分布式结构爬虫通过一台特定的链接服务器（Master）对爬虫抓取过程中产生的链接集合进行维护，各个爬虫节点通过不断地向链接服务器获取链接并向网络发起下载请求。然后将新采集的链接列表页传输给链接服务器，链接服务器还会根据各个爬虫节点的负载情况，进行负载分析，调整爬虫资源，如图 6-1 所示。

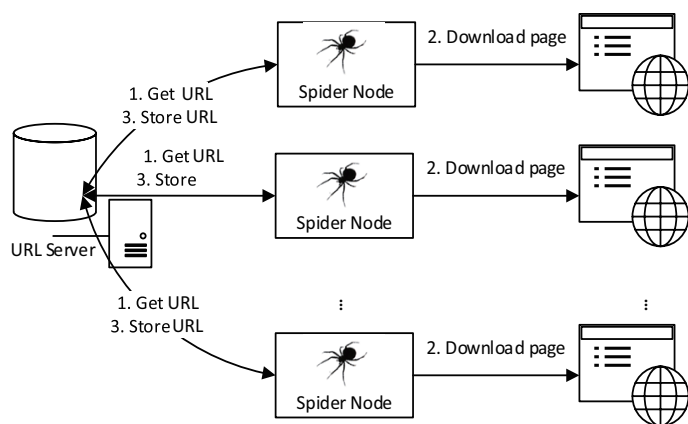


图 6-1 主从分布式结构爬虫示意

主从分布式结构的爬虫，层次结构清晰，拥有灵活的 URL 分发策略，但是随着爬虫节点的大规模增加，链接服务器作为整个数据抓取系统的管理者，容易成为系统瓶颈，存在单节点故障，甚至有可能导致系统崩溃。

6.1.2 对等分布式结构爬虫

对等分布式结构爬虫相对于主从分布式结构爬虫，减少了链接服务器，服务器集群中所有的服务器的任务均是进行数据抓取。与主从分布式结构爬虫的不同点是：每个爬虫节点在获得数据之后，提取出的新 URL 集合通过一定的规则（如通过哈希取模方式），将新链接分别发送给对应的爬虫服务器，每个爬虫服务器都是按照相同的规则获取链接。例如，集群中存在 10 台爬虫服务器，

对链接取哈希值计算，利用相应哈希值对 10 求余，根据余数确定服务器编号，以达到爬虫内部之间相互交换链接，如图 6-2 所示。

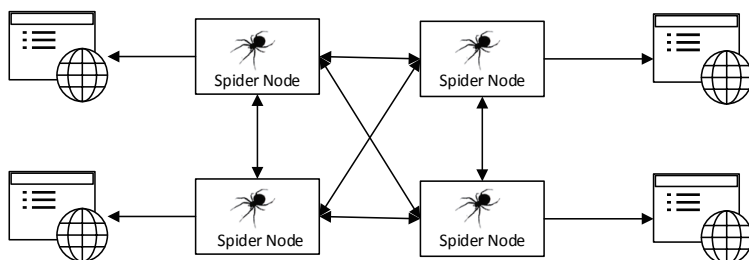


图 6-2 对等分布式结构爬虫示意

对等分布式结构爬虫中，每台服务器都可以单独工作。但是容易出现的问题是：可能会导致某台服务器的任务过重，负载不均衡，倘若其中任何一台服务器出现故障，那么这台服务器对应的链接可能会无法进行抓取。但也有通过一致性环形哈希确定链接对应的爬虫节点，不再通过简单的哈希取模的方式选择服务器，保证在服务器即使出现故障的情况下，也能正常进行数据抓取工作。除此之外，对等分布式爬虫还具备高可扩展性、容错性。但是由于服务器之间的频繁相互通信，会占用部分宽带资源。

6.1.3 基于分布式计算平台爬虫

在大数据和云计算时代之前，从工程实践角度来看，需要关心爬虫之间的通信，且代码复用程度低，爬虫逻辑和通信机制会同时考虑，而在进行其他分布式工作中则需要重写相关通信逻辑。利用分布式计算平台，则无须过度关注爬虫的分布式通信和任务分配，仅需将工作重点集中在爬虫逻辑。爬虫的实际处理过程是一个工作流的处理，同 Storm 的流式处理机制类似，并且可以将对等网络中的爬虫内部任务进行拆解，还可以根据任务的繁重程度，进行调整，最大限度地利用系统资源。基于 Storm 的分布式爬虫基本逻辑示意如图 6-3 所示。

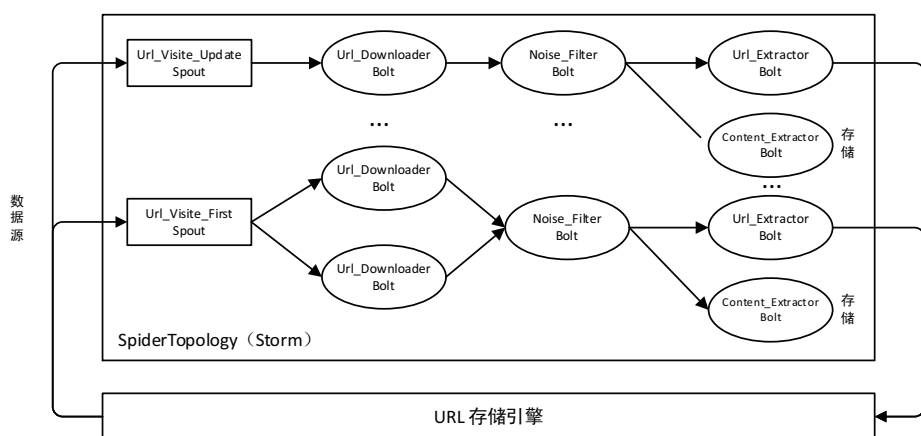


图 6-3 基于 Storm 的分布式爬虫基本逻辑示意

同所有的爬虫工作流程类似。首先抓取数据；其次对数据进行噪声去重；然后进行链接提取和网页内容提取，Spout 进行源源不断的数据输入，Bolt 不断地进行数据处理。

利用 Storm 的分布式计算，实质上是爬虫的主从结构和对等结构的混合体，能够对处理节点进行动态分配，随时扩展工作节点数量，不存在主从结构中的单点故障，也满足 URL 分配的灵活控制。除此之外还有一个最大不同点在于更加细粒度的任务分配方式，将网页下载、网页去噪、链接提取等拆解到更多不同的机器中并行工作，不管是以往的主从结构还是对等结构，每个服务器节点的工作都是基于进程级别，即使是多线程情况下执行任务，效果也不如此种方式。此外，还可以通过 Storm UI 进行瓶颈分析，从而进行策略调整，减少在分布式协同工作中的工作量，分布式计算框架与爬虫逻辑实现高内聚低耦合。

6.2 网页解析

网页解析是爬虫下载过程中最重要的任务，是爬虫的核心功能之一，包含对请求状态码处理、网页链接的处理、网页噪声处理、内容提取、链接提取等。

6.2.1 状态码处理

在与网站服务器进行数据访问时，并不一定会返回结果，需要根据 HTTP Header 中状态码的信息进行分析，这些状态码如表 6-1 所示。

表 6-1 状态码含义示意

状态码	示 例	含 义
1xx	100 (Continue)、101 (Switching Protocols)、102 (Processing)	代表请求已经被接受，但是需要继续处理
2xx	200 (OK)、202 (Accepted)、204 (No Content)	服务器已经成功接收处理
3xx	300 (Multiple Choices)、301 (Moved Permanently)、304 (Not Modified)	表示重定向，客户端需要进一步操作
4xx	400 (Bad Request)、401 (Unauthorized)、403 (Forbidden)、	表示客户端的操作请求，服务器无法完成，客户端需要检查操作的正确性
5xx	500 (Internal Server Error)、501 (Not Implemented)、502 (Bad Gateway) ...	由于服务端原因，无法正常完成请求

爬虫需要根据服务器返回的状态码给出相应的反应，在 4xx 状态码中无法直接获取数据，相应的访问链接需要标记为无效链接。针对 5xx 链接，需要对标识为可以继续尝试的链接，在下次更新策略启动时，再次进行尝试。

6.2.2 链接去重

爬虫每一次进行页面分析都会获得新的链接集合，然后这些链接集合难免存在重复情况。不仅仅是在当前存在页面重复，更有可能在整个分布式系统中存在页面重复。链接去重的目的是提升数据采集准确率、避免重复的工作。在过去采用哈希过滤、分布式数据库、磁盘路径方法试图解决链接去重问题。

(1) 哈希过滤。最直观的做法是将链接集合放入内存，然后利用哈希表进行过滤。但是在分布式爬虫系统中，采集的链接历史记录超过十亿，若每个链接拥有 100 字符，每个字符占用 2 字节内存，则十亿的 URL 集合占用内存超



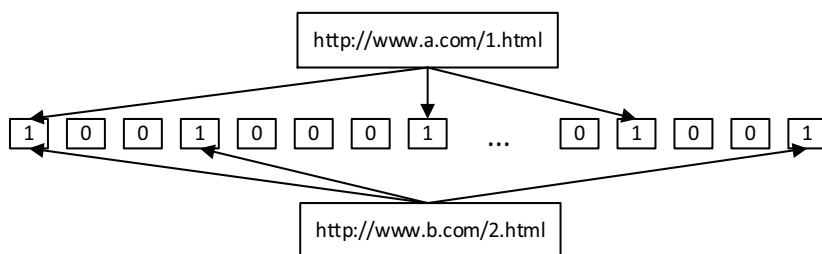
过 200 GB，通过内存的哈希过滤显然是不可取的方式。

(2) 分布式数据库。若将这十亿的连接存放在整个庞大的分布式系统中，利用分布式数据库区解决此问题，虽然可以解决重复问题，但是分布式数据库的性能问题将会凸显出来，分布式爬虫的整体效率将会下降。

(3) 磁盘路径方法。在磁盘路径去重的方法中，首先将获得的链接进行 MD5 编码，得到 32 位长的 MD5 值，每个字符表示一个路径节点，32 位的 MD5 则是一个 32 位的存储路径；然后将链接的 MD5 值映射到磁盘的存储路径中；最后通过 MD5 路径判断链接是否存在，不存在则创建相应路径，存在则过滤。通过磁盘路径法，对于中小数据集的链接是可取的，但是对于十亿级别的链接集合，最终会因为创建的文件碎片过大，极易导致文件管理系统崩溃。

综上所述，缺陷均比较明显：哈希过滤占用内存过大，分布式数据库处理太缓慢，磁盘路径方法使得文件碎片过大。因此，对于如何进行高性能、高效率的链接去重，变成一个强大的挑战。采用布隆过滤器（Bloom Filter）的方式可以有效地弥补上述三种方式中的不足。

布隆过滤器常用于检测元素在一个集合中的存在情况，它的时间复杂度和空间复杂度都优于其他方式。布隆过滤器的主要思想是：首先当一个链接被放置到链接集合时，通过 k 个不同哈希函数将这个元素拆分为 k 个值；其次利用映射函数，将 k 个值分别映射成一个位数组（byte[]）中的 k 个点；然后将所在位置全部置为 1，查询时，通过相同的 k 个哈希函数和映射函数；最终确定在位数组上的所有值是否为 1 即可，倘若这些点中存在一个或者多个 0，则表示该 URL 一定不存在。例如，设定 $k=3$ ，即需要三个不同的哈希函数，将对应的链接映射到位数组的三个位置中，映射方式如图 6-4 所示，在映射后的数组中，如果三个即将映射的值都已经被置为 1，则可认为已经存在。

图 6-4 在 $k=3$ 时的布隆过滤器映射方式示例

对于 k 值的选择（哈希函数的个数），设定 m 为该位数组的长度， n 为大致插入的元素个数，则 $k=(m/n)\ln 2$ 。例如， m 为百亿字节， n 为大约 10 亿网页，则 $k \approx 7$ 。而当 m 为百亿字节时，占用内存约 10 GB，对于一台内存为 64 GB 的服务器，足足够用。

布隆过滤器的时间复杂度为 $O(k)$ ，而基于传统的链表、树及哈希表的方式，查询时间复杂度分别为 $O(n)$ 、 $O(\log n)$ 和 $O(n/k)$ 。但是布隆过滤器也依然存在缺陷，随着链接集合的不断壮大，误算率也会逐步增加，布隆过滤器绝不会遗漏任何一个已经存在的链接，但是它有一定概率将不存在的链接判定为存在。被判定的链接占有的地址位，有可能已经被别的链接设定为“1”。

6.2.3 广告识别

网页中存在大量噪声，噪声信息包括头部导航栏、侧边栏、广告信息等，导航栏和侧边栏信息可以通过正文提取时，根据文字分布情况进行过滤，但是广告信息复杂多变，在正文中也存在广告信息的内容。进行有效的广告信息识别，对于准确提取文档内容有极大的帮助。

常见的广告信息包含广告链接、广告文字、广告图片、Flash 动画等。从网民浏览网页的角度看，某些广告严重影响了浏览体验；从搜索引擎角度看，严重影响了网页提纯，对搜索结果、网页相关性排序产生了一定的干扰。广告



识别的方式方法有很多种，例如，通过网页结构分析、机器学习分析广告特征等方式。但是在工程领域这样很难达到很好的效果，一方面是处于不断变化中的广告具备多样性，网页结构也在不断改变；另一方面通过上述方式识别率很难得到保障。在实际中，采用 Adblock plus 的思想对广告进行拦截，Adblock plus 是著名的广告过滤插件，安装在浏览器中，对浏览器打开的页面进行广告过滤，达到了非常好的过滤效果。Adblock plus 对于搜索引擎在对广告范围定义和广告鉴别方式有很好的借鉴作用。

(1) 广告范围定义。Adblock plus 认为互联网中的广告并不是所有广告都需要过滤，而是过滤用户常常访问的网站的广告。

(2) 广告鉴别方式。判定页面的元素是不是广告，并不需要通过分析网页结构、甚至机器学习的方式，并采用众包的思想。如果大家都浏览过这些页面，而且都判定这些页面中某元素是广告，那么 Adblock plus 则认为这就是广告。

Adblock plus 利用众包和大众化的思想将互联网主流站点网页中的广告几乎过滤到了极致，它通过网民反馈回去的广告特征，构建了一套完整的广告过滤规则库，这些规则库每周都在不断更新，能够适应广告的不断变化，占用内存小，处理速度极快，爬虫也将利用这些规则库进行页内广告元素移除。广告拦截的规则信息如下所示：

```
/adflash/*
/attachments/ad/*
/tan1.js
@@||hichannel.hinet.net^$object-subrequest
@@||poster.weather.com.cn/p_files/base/$image
@@||union.bokecc.com/crossdomain.xml
hk.search.auctions.yahoo.com###yauadysmh
hk.yahoo.com###mntl1
|114lm.com^$third-party
|116b.com^$third-party
##div#*ads*
```

广告拦截使用上述规则，对网页中的链接、元素 ID 等进行分析过滤，而搜索引擎利用规则进行元素移除，例如，针对规则“`||114lm.com^$third-party`”表示阻挡站点 114lm.com 中的第三方请求，则第三方链接将会全部移除，包括图片、Iframe、Flash 等。“`##div#*ads*`”表示移除（Adblock plus 采用隐藏）所有 DIV 中名称 Id 中包含 ads 的 DIV 元素。Adblock plus 这样的规则累计约有 6 万条。针对目前大众用户访问的主流站点，如果将每一个规则和网页内的每一个元素进行比较，则可能会导致性能问题，而实际上灵活使用这些规则将会达到较高的处理性能，主要从规则转换和查找表两方面着手。

(1) 规则转换。将所有的规则全部转换为程序可以识别的正则表达式，例如，规则“`ad*show.avi`”将会被替换为正则表达式“`/ad.*show\avi$/`”。

(2) 建立查找表。查找表是一个哈希表，哈希表的键是广告拦截规则的特征标识，选取的特征标识为任意连续的 3 个英文字母，例如，“`ad*show.avi`”可以选择“`sho`”为其特征，对每个规则都进行特征标识抽取，然后将特征相同的规则，转变为正则表达式放入链表集合，哈希表的值则是这些链表的集合。比如对页内名为“`ad_show.avi`”的元素进行匹配时，同广告拦截规则一样获取连续的 3 个英文字母作为其特征，不同的是需要从第一个英文字母开始，依次取多次，需要取出“`sho`”“`how`”“`avi`”三个，然后分别在哈希表中根据键查找出相应的正则表达式集合，进行匹配，如图 6-5 所示。

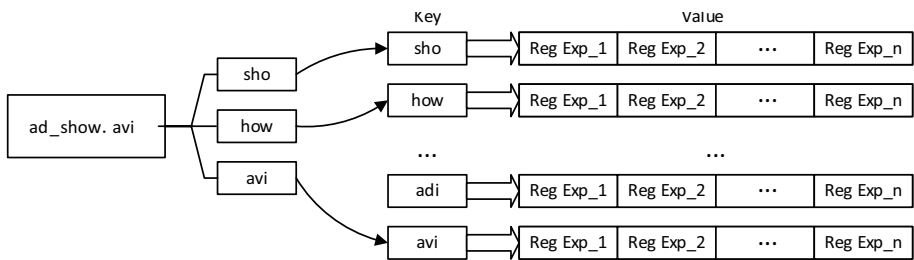


图 6-5 元素利用查找表获取正则表达式示意



在匹配过程中，对通过查找表获得的所有正则表达式集合进行一一匹配，一旦命中则不再继续向下匹配。

在搜索引擎中使用广告过滤，除对数据提纯达到了很好的效果，对于广告过滤后的网页存储、网页快照等方面也有着积极作用。

(1) 网页存储方面。因为已经对文件中的广告相关数据进行了移除，会减少文件存储大小，节省相应存储空间。

(2) 网页快照方面。移除广告相关资源、链接加载之后，网页快照显示速度加快，网页呈现也会无广告，用户体验有所提升。

(3) 用户体验方面。对于图片搜索、视频搜索也可以减少其中的广告图片数量和广告视频数量，对用户体验有着积极辅助作用。

6.2.4 网站地图

网站地图是网站站长标识出网站导航的信息，存在 HTML 类型和 XML 类型。

(1) HTML 类型。主要针对网站用户使用，便于用户查找需要的信息或子站点。

(2) XML 类型。主要供搜索引擎访问参考地图，网站之所以会自动提供一份站点地图给搜索引擎，一方面是因为该网站含有大量的动态内容；另一方面是网站内存在大量的网页，但是内容页之间并没有直接相连，存在一些孤立网页，通过网站地图将其关联起来。例如，Google 给自己提供的网站地图如下所示：

```
<urlset xmlns="https://www.sitemaps.org/schemas/sitemap/0.9" xmlns:mobile="https://www.google.com/schemas/sitemap-mobile/1.0">
<url>
<loc>https://www.google.com/</loc>
</url>
<url>
```



```
<loc>https://www.google.com/3dwh_dmca.html</loc>
</url>
...
</urlset>
```

爬虫通过分析网站地图，对网站进行深入抓取。除此之外，网站地图也是搜索结果中站点的层次描述信息的依据之一。

6.2.5 非网页数据获取

非网页数据中，主要包括 PDF、Word、PPT、Excel、TXT，除 TXT 可以直接进行数据读取之外，PDF、Word 和 PPT 都有相应的读取引擎进行内容获取，如表 6-2 所示。

表 6-2 各文档类型对应读取引擎

文档类型	读取引擎
PDF	Apache PDFBox、xpdf
Word、Excel、PPT	Apache POI

除此之外还包括 RSS（Really Simple Syndication），为了增强爬虫数据的时效性，除加快爬行速度之外，还需要控制实时数据的抓取，而最佳的实时数据即是博客、新闻网站。大部分新闻网站都提供 RSS。RSS 是一种消息的规范格式，经常发布信息更新的网站会通过 RSS 使得用户订阅新数据，使用这些 RSS 数据，具备信息高度准确、高效率、高实时的特点，作为搜索引擎，非常重视实时信息源。

RSS 的解析方式同 HTML 类似，同 XML 一致，且有固定格式，通过分析 XML，可以精准提取出文档标题、链接、作者、发布日期、摘要描述等相关信息，与通过 HTML 分析得到的结果数据类似，不同点在于 RSS 提取精准度更高。

除对 PDF、Word、PPT、Excel、TXT、RSS 抓取之外，还会对图片数据进行抓取。



6.2.6 网页去重

爬虫对数据的抓取并不是每个网页都会被存储，网页将会被进行重复性过滤。虽然通过 URL 不重复解决了部分问题，但是难免存在相似的搜索页面，例如，某篇博客，被转载到另外一个网站，甚至一个网站的多个域名，但是都被某关键词搜索出来。存储重复性文档及索引重复性文档，对服务器资源包括存储空间和 CPU 消耗，都是额外且不必要的任务。对于爬虫获取的网页是否具有相似性的三种情况分别是完全相似、内容相似和局部相似。

(1) 完全相似。文档之间不仅仅在内容上一致，在网页布局格式上也一致，此类完全重复页面一般属于一个站点多个域名。

(2) 内容相似。文档之间内容相同，但是网页布局格式不同，此类内容重复页面一般发生在转载。

(3) 局部相似。对于文本中的内容，具有部分相似的可能性，此类情况可能发生在文章引用。

按照传统的文档相似度计算，例如 Jaccard 相似性。对网页信息中的词汇进行分词处理，然后通过 Jaccard 相似性计算方法，公式如下所示：

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

A 和 B 分别代表两个网页文档， $J(A,B)$ 表示两个文档拥有相同的词汇数量占有两个文档一共词汇数量的比例，当达到一定比例，即可认为是相似文档，进行忽略。

如果通过 Jaccard 相似性等普通的方法计算文档相似度，仅仅存在较小的可能性，因为在海量数据下，很难体现出实质意义。若是利用传统的哈希算法进行去重，也只能去除内容完全相同的文档，文档中任何一个字不一样，都会

导致文档之间计算的哈希值相差很大。但是传统的哈希算法给予新的思考，若每个文档都拥有一个特殊的哈希值，哈希值越相近，则认为文档相似程度越高。通过新的思考，赋予文档以文档指纹的概念，指纹值越相近，相似度越高，这即是 Simhash 算法。

Simhash 的思想是，将输入作为一个二维集合，集合中包含 N 个字符串及字符串对应的权重，输出的是一个 M 位的二进制签名串。首先，它将每个字符串进行字符串二进制化；其次对字符串的二进制值进行加权；然后将所有字符串的加权字符串进行累加，累加顺序按照各自位置进行，累加之后进行降维；最终得到签名的二进制字符串。

Simhash 充分利用了降维的思想，将高维度的特征矢量映射成低维度的特征矢量。Simhash 的降维过程是将一个复杂的信息表示成一个长度为 32 位或者 64 位的二进制字符串，正如居民身份证一样，每个人都有庞大的信息，但是唯一的标识码则是身份证号，在信息系统中，比起用个人的特征（如身高、体重等）去寻找一个人，不如用身份证号迅速，降维的目的也在于此。最终通过海明距离进行相似判断。海明距离是计算两个字符串对应位置的不同字符个数。通过示例说明，对于“香港警察逮捕罪犯”计算 Simhash 文档签名。

（1）分词，权重计算。过滤清晰数据，对文本进行分词处理，计算每个词的相应权重，如表 6-3 所示。

表 6-3 “香港警察逮捕罪犯”文本分词与权重计算

关键词	权 重
香港	3
警察	4
逮捕	2
罪犯	4

（2）词语二进制化。将分词后产生的所有词汇，进行二进制化处理，如表 6-4 所示。在实际中，计算字符串的二进制值长度与字符串长度有关。

表 6-4 “香港警察逮捕罪犯”分词后字符串二进制化

关键词	权 重	字符串二进制化
香港	3	100110
警察	4	010010
逮捕	2	110011
罪犯	4	101010

(3) 二进制加权。将字符串二进制化后的值，进行加权。将权重与二进制对应的每一位相乘，字符串中的“0”视为“-1”，如表 6-5 所示。

表 6-5 “香港警察逮捕罪犯”字符串二进制加权

关键词	权 重	字符串二进制化	加权值
香港	3	100110	3-3-3 3 3 -3
警察	4	010010	-44-4-44-4
逮捕	2	110011	2 2 -2 -2 22
罪犯	4	101010	4-4 4-4 4-4

(4) 累计合并求解。将四个词中相同位置的加权值累计求和，最终得到“57-5-713-9”。

(5) 降维二进制化。通过约定，若累计合并的值中，大于 0 的位设定为 1，小于等于 0 的位设定为 0，因此最终“香港警察逮捕罪犯”的 Simhash 签名为“110010”。

通过上述过程，完成对“香港警察逮捕罪犯”文本信息的文档指纹签名。同理若是“人工智能未来 10 年内将超越人类”的指纹签名为“001100”，则需要通过海明距离（Hamming Distance）进行计算两者相似度。对于“001100”和“110010”的海明距离，是将二者异或之后得到的二进制字符串中“1”的个数，因此“人工智能未来 10 年内将超越人类”与“香港警察逮捕罪犯”二者海明距离为 5。

通过海明距离已经很方便地进行了数据相似度计算，但是将其扩展到海量的数据中进行重复性验证还不能够满足，算法性能很快，但是随着文档数量的

不断增加，处理性能也会逐步减弱，与文档数量呈现线性正比例关系。

根据抽屉原理，将 N 个物品放置到 M 个抽屉中时，至少有一个抽屉容纳 $\lceil N/M \rceil$ 个物品（ $\lceil N/M \rceil$ 表示向上取整）。也就是若把 10 本图书放置到 9 个抽屉里面，至少有一个抽屉容纳两本图书（ $\lceil 10/9 \rceil = 2$ ）。基于抽屉原理的思想，如果 Simhash 的签名结果是 64 位的二进制字符串，设定当海明距离小于或等于 3 时，则视为文本相似，可以去重。将 64 位的二进制字符串等分为 4 份，若两个文档相似，则 4 份二进制字符串中，至少有一份是一样的。基于此，建立简单的哈希查找表，其中键为 16 位的二进制字符串，值为指针，指向完整的二进制字符串，存指针的目的是防止数据被多次存储，占用存储空间，如图 6-6 所示。

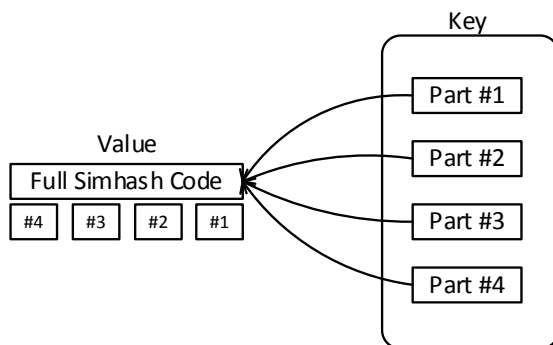


图 6-6 Simhash 比较过程中的哈希查找表

在爬虫进行分析处理过程中，为减少计算量，并不是将文档中的所有正文进行文档指纹签名，在获得文档的正文内容之后，按照如下步骤进行：

（1）关键词权重计算。对正文内容分词，对正文进行关键词权重分析，将正文的关键词按照权重排序。

（2）依据关键词计算文档指纹签名。筛选前 N 个关键词，进行分析文档指纹签名，最终得到文档的文档指纹。

（3）建立哈希查找表。将文档的指纹签名等分为若干份，建立哈希查找表。



(4) 查找比较。将比较的文档的指纹签名进行拆分,通过哈希表按块查找。将对应集合依次比较,判定是否已经被收录。

通过 Simhash 降维的方式是大数据下数据分析处理的常用方法。假设搜索引擎现在已经存在 10 亿(约 2^{30}) 文档,按照传统的文档相似度计算方法,则需要进行 10 亿次比较,而通过 Simhash 算法之后,只需要 6 万多次比较,比较次数大大减少,提升爬虫数据抓取的性能。

从另外一个角度看,网页重复或高度相似并不是一件完全负面的情况。虽然对用户来说,搜索出来的页面太多相似可能导致发现不了关键信息,但是从搜索引擎的角度看,网页重复率越高,则说明文档内容受欢迎程度越高,也预示着文档相对其他文档比较重要,在搜索结果中,应赋予一定较高评分,尤其是文档的原创页面。

对于完全重复的文档,可以采用删除的方式,对于相似文档,则通过对文档进行归类分组。Simhash 的意义不仅仅在于防止互联网上内容相似的文档,还在于对爬虫的二次更新数据抽取也奠定了去重基础。

6.2.7 链接提取

链接的提取方式有两种:一种是采用 HTML 解析器,提取网页中的所有“a”标签;另外一种采用正则表达式。采用正则表达式提取链接是一种比较简单可行的方式。对于一个链接“资讯”需要提取两部分内容,一个是“href”中对应的链接,另一个是“a”标签中的链接描述信息。

但是,对于链接提取存在这样的情况,对“http://www.site.com/news”页面进行分析后抽取出的“a”标签如下所示:

```
<a href="a.html" target="_blank">五中全会后深改组首开会通过8项文件</a>  
<a href="/b.html" target="_blank">中美为战略伙伴而竞争</a>  
<a href="./c.html" target="_blank">五一劳动奖表彰大会在京举行</a>  
<a href="../d.html" target="_blank">国务院出台16条措施防止国资流失</a>
```

在上述标签中，提取到的“href”值为相对链接。需要将相对链接转换为绝对链接，针对访问的基链接（base-url）为“http://www.site.com/news”，因此各相对链接转换后的绝对链接如表 6-6 所示。

表 6-6 相对链接转换为绝对链接示例

相对链接	含义	绝对链接
"a.html"	当前链接	http://www.site.com/news/a.html
"/b.html"	Web 站点的根目录下网页	http://www.site.com/b.html
"./c.html"	当前目录下链接	http://www.site.com/news/c.html
"../d.html"	上级目录下链接	http://www.site.com/d.html

因此在提取链接时，需要先验证链接是相对链接还是绝对链接。若相对链接需要转变为绝对链接，则绝对链接保持不变。但是也不能直接通过验证字符串以“http”开头进行验证，互联网中的数据除“http”“https”访问协议，还可能以“FTP”等其他方式访问。

6.2.8 爬虫协议

一个友好的搜索引擎爬虫，需要遵守站长制定的爬虫协议，尊重网站本身的数据信息。爬虫协议中约定了网站中哪部分数据被收录，哪部分不允许被收录，以及哪些爬虫不允许授权访问数据等，因此全称为“网络爬虫排除协议”。正如去一个朋友家做客，朋友家的阳台和客厅你可以访问，但是想去朋友家卧室看看时，需要征求朋友的意见。

爬虫协议内容为一个 robots.txt，约定在网站的根目录下，包括一级域名和二级域名，某网站的 robots.txt 如下所示：

```
User-agent: *  
Disallow:
```

```
Disallow: /cgi-bin/  
Disallow: /home/  
Disallow: /phoenixtv/  
Allow: /
```

(1) User-Agent。代表允许的爬虫名称，* 表示所有爬虫有效，各大搜索引擎爬虫名称如表 6-7 所示。

表 6-7 各大搜索引擎爬虫名称

厂 家	爬虫名称
百度	Baiduspider
Google	Googlebot、Googlebot-Mobile
搜狗	Sogou spider
有道	YodaoBot
好搜	360Spider
必应	bingbot

(2) Disallow。/cgi-bin/ 表示不允许访问当前域名下的 /cgi-bin/ 目录下所有资源。

(3) Allow。表示允许的位置或者目录。

解析爬虫协议的，需要分析针对当前爬虫的 Disallow 集合和 Allow 集合，只需要在域名上做一次比较。对于搜索引擎，需要准确地识别各大网站针对爬虫名称作出的爬虫约定。但是对于刚刚研发完毕或者正在研发的搜索引擎爬虫，各大网站也许不会针对其作出特定约定，则需要遵从 Useragent 为“*”的约定。若网站并未设置 robots.txt，则按照默认允许可以访问处理。

网站之所以设定 robots.txt，也是基于其保护网站本身用户隐私的需要，保障用户隐私不被侵犯。作为爬虫，需要尊重网站的意愿并帮助其保护用户隐私。基于此，爬虫除遵守爬虫协议之外，还应注意避免在白天高峰期抓取网站，给网站流量带来负荷。

6.3 网页结构化

爬虫通过对网页分析，将网页从非结构化数据解析为结构化数据信息。其中包括网页的编码信息、网页的正文信息、网站的标题等。

6.3.1 网页的编码信息

网页的编码信息是后续其他工作中非常重要的基础数据。网页开发者在设定网页展示编码的时候，会根据实际情况有所不同，例如，ASP.NET 默认采用 GB 2312 的方式，但是 JSP 国内开发者习惯性使用 GBK 或者 UTF-8 编码。国外网站中，由于语言不同，网页的编码也不尽相同。因此，正确解析编码，对页面数据处理至关重要，处理不当就会形成乱码。

一般网页的 HTML 代码中对编码都有相关的说明，大部分在 HTML 的“meta”标记中。“meta”表示页面的元信息，也就是最基本的信息。“meta”信息对网站开发者来说，可以不存在，但是对于标准网页或者高质量的网页，拥有“meta”信息将会帮助站长很好地进行搜索引擎排序，对网站的搜索引擎优化就包含此部分信息，如下所示：

```
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
```

上述表示当前请求的文件类型为文本 HTML 数据，字符编码采用 GB 2312。GB 2312 是指《信息交换用汉字编码字符集》，它是由中国国家标准总局 1980 年发布的字符交换编码，适用于汉字处理、汉字通信等系统之间的信息交换。当然，字符编码不仅仅局限于这一种表示方法，还包括其他编码方式，如下所示：

```
<meta charset="UTF-8">
```



```
<meta http-equiv="content-type" content="text/html; charset=gbk"/><meta content="text/html; charset=gb2312" http-equiv="Content-Type">
```

6.3.2 网页的正文信息

网页的正文抽取不能简单地理解为将网页中的文字提取出来，文字的提取仅仅是对网页所有信息的抽取，而正文抽取是将网页中本身需要表达的内容提取。例如，在打开新闻页面时，最重要的信息是新闻的内容，与其他导航栏、侧边栏、相关链接及广告都无关。这些噪声信息在正文提取过程中需要进行噪声过滤。从搜索引擎角度分析，如果搜索结果的显示内容摘要正文相关信息，不包含无意义的导航栏、侧边栏等其他信息，则将会对用户体验有很好的辅助作用。此外，对网页正文实现精确抽取，对网页文本进行准确分类、网页相似度分析、搜索用户个性化推荐也起到非常重要的作用。

网页正文提取有很多种方法，包括 HTML DOM 树分析的方法、基于统计学习的方法等。这些方法在实际应用中，都有非常大的难度，尤其是性能问题。对于统计学习的方法，语料库要求比较高，若要达到较高的准确率，前期人工参与程度较大，并且不一定能够适应因网页结构的改变导致的网页特征改变。因此，试图寻找网页正文提取算法在效果和性能方面有一个较为优秀的方案。

从网页开发者的角度和站长对信息公示的角度分析，网页正文信息从视觉上具有聚集性和密集型两大特征。

- (1) 聚集性。网页正文基本上都是在某一特定区域，具有聚集性。
- (2) 密集型。网页正文一般占有整个页面的版面相对较大，具有密集性。

根据网页在视觉上的聚集性和密集型，对网页的 HTML 代码进行两大特征分析。以各大新闻门户网站进行分析，以行号为 x 轴，以所在行文本长度作为 y 轴。以新浪某新闻网页文本分布为示例，如图 6-7 所示。

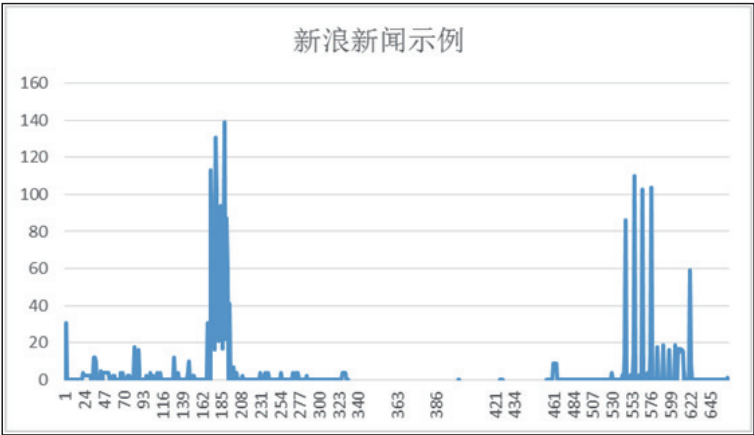


图 6-7 新浪某新闻网页文本分布示例

观察图 6-7 可知,最具有密集性和聚集性的是 162~210 行的区域。520~630 行的区域,虽然聚集但是不够密集,很有可能是相关性文章或者文章评论信息。再以搜狐某新闻网页文本分布为示例,如图 6-8 所示。

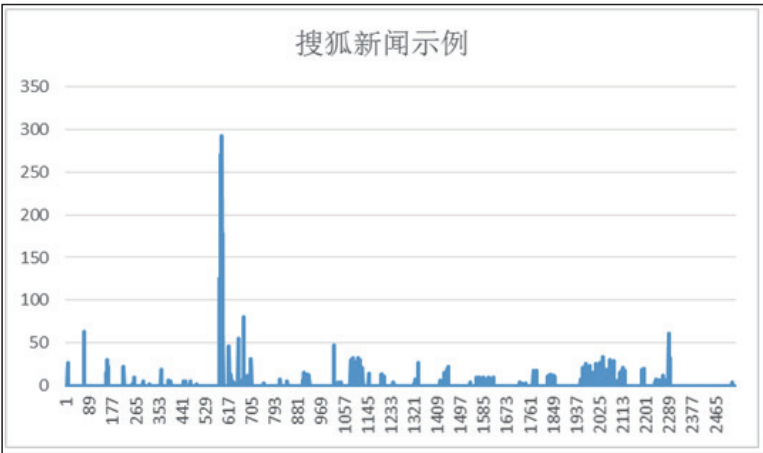


图 6-8 搜狐某新闻网页文本分布示例

观察图 6-8 可知,网页密集区和集聚区大致在 580~610 行,该区域属于正文区域。同理,对网易新闻页面及腾讯新闻页面文本分布进行分析,如图 6-9 所示。

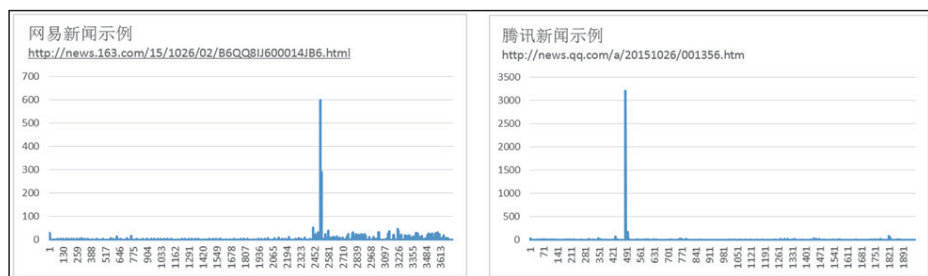


图 6-9 网易新闻及腾讯新闻页面文本分布示例

观察图 6-7、图 6-8、图 6-9 可知，可将网页正文抽取从视觉分布转变为求数据最大分布的问题。针对博客类网站依然会有类似效果，博客园与推酷网的博客文本分布情况如图 6-10 所示。

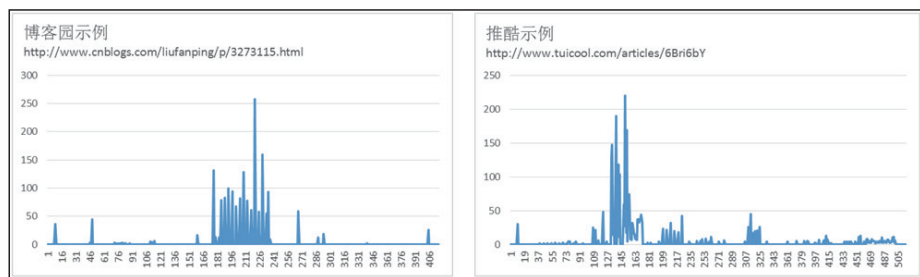


图 6-10 博客园与推酷网的博客文本分布情况

基于聚集性和密集性特征的网页正文提取，可以解决大部分新闻、博客、等文章类的网站，但是不能解决论坛、网站首页等非文章类页面的抽取，图 6-11 所示为新浪与网易首页文本分布情况。

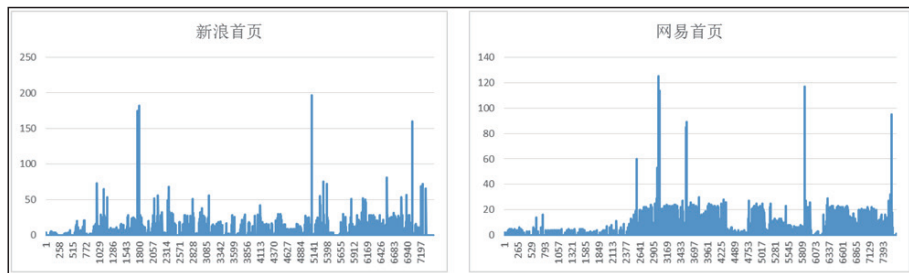


图 6-11 新浪与网易首页文本分布情况

观察图 6-11 可知，两者的文本分布比较均匀，因此会给抽取带来一定的难度，主要是由于网页本身的针对性均不明确，更多的是信息引导。对于一级域名及二级域名，需要先分析 HTML 标签的自描述，即“meta”标签下“name”为“description”的信息。一级域名的 www.ifeng.com 的自描述信息如下。

```
<meta name="description" content="凤凰网是中国领先的综合门户网站，提供含文图音视频的全方位综合新闻资讯、深度访谈、观点评论、财经产品、互动应用、分享社区等服务，同时与凤凰无线、凤凰宽频形成三屏联动，为全球主流华人提供互联网、无线通信、电视网三网融合无缝衔接的新媒体优质体验。"/>
对于二级域名的“sports.ifeng.com”自描述信息如下。
<meta name="description" content="凤凰网体育频道提供文字、图片、视频、数据、新闻等主流体育资讯服务，以凤凰网媒体价值观深度挖掘体坛资讯背后故事，以独特的视角报道全球焦点赛事。" />
```

除一级域名和二级域名之外的其他页面，很少包含自描述信息。自描述信息的主要目的用于准确地提供网站摘要描述信息。这也是用户在使用各大搜索引擎时，能够搜索出关系网页的介绍信息的原因。网页解析过程中，可以抽取此部分信息，为最佳正文内容。在最差的情况下，网页解析可以通过 HTML 代码的“body”内容获取文本信息。

在文本获得之后，可能会存在这样的情况，文本内容类似如“中国首试特大航天群伞 重型直升机出动(图)”的乱码信息，实质上并不是，它是文字“中国首试特大航天群伞重型直升机出动(图)”的 UTF-8 编码形式，类似还有“%E4%B8%AD%E5%9B%BD”实质是“中国”的 ASCII 字符，因此对于部分文本的解码是非常必要的。

综上信息，在 HTML 正文抽取过程中，首先需要获取“meta”信息下面的“description”内容；在获取无果的情况下，再通过网页文本的聚集性和密集性获取正文内容；在上面两种情况均无果的情况下，再采用正则表达式提取网页文本信息。



6.3.3 网站的关键词信息

同自描述信息一样,在 HTML 的“meta”标签中有一项“name”为“keyword”,某网页中标识如下:

```
<meta name="keywords" content="开发者,博客园,开发者,程序猿,程序媛,极客,编程,代码,开源,IT网站,Developer,Programmer,Coder,Geek,技术社区" />
```

与自描述信息不同的是,关键词信息在任何页面都有可能存在。当页面不存在“keywords”的元信息时,则按照 3.6 节在文本中进行抽取。关键词信息的抽取,主要是为确定文章主题、文本分类及搜索排序提供一个参考。

6.3.4 网站的标题

在一般情况下,网站的标题,通过正则表达式直接抽取出 HTML 标签中的“<title>”数据即可,但是要做到网页标题的精准抽取却比较难,例如下面示例。

```
<title>重庆人力资源和社会保障网——[北碚]全面发力助推农民工返乡就业创业</title>
<title>聚焦“十三五”:新时期我国经济社会发展路径认识上的新突破_滚动_新闻_中国政府网</title>
<title>黄河流域2015年水土保持监测工作会议在河南洛阳召开</title>
```

在上述示例的 title 标签中,都是实际存在的标题,但不足的是夹杂着网站的名称,并且网站名称在标签中的位置不固定。因此网站标题不仅仅通过正则表达式抽取,还需要其他辅助措施。

(1) 基于元数据获取。通过 HTML 元数据中关于“title”描述的值获得,是一种非常精确的获得方式。如下所示,但并不是所有的网页都会通过“meta”进行标识。

```
<meta property="og:title" content="国务院出台16条措施防止国资流失" />
```

(2) 基于链接描述获取。一个页面基本上是通过一个链接打开,而打开这

个链接的过程却又是通过前一个链接中的文本。因此，上一个链接中的文本即是对该链接的描述，如果大部分都描述一致，则说明这部分文字很可能是对应链接的标题或者中心思想。因此链接的价值不仅仅在于链接本身，还有链接出去的文字，如下链接示例：

```
<a href="http://news.a.com/mainland/special/xmh/" target="_blank">
地方一季度GDP数据开门红</a>
<a href="http://news.a.com/a/20151107/46153709_0.shtml" target="_
blank">李克强经济形势公开课(第三季)</a>
<a href="http://www.cnblogs.com/liufanping/p/4489864.html">开源搜索
引擎Iveely 0.8.0发布，终见天日</a>
```

换一个角度，标题是人们去点击这个链接的依据，同理，搜索引擎搜索结果中的标题，也将会成为用户是否点击搜索结果的依据，保持链接在互联网中本身的文本含义，有助于用户对网页的理解。

（3）基于正文匹配方式。第一步根据网页的正文提取完毕之后，也提取出了网页源码中关于“title”标签中的标题“老程序员应该记住的 5 件事 _IT 新闻”，提取的正文如表 6-8 所示。

表 6-8 某网页正文抽取内容

行 号	行对应内容
1	IT 新闻 » 程序员
2	老程序员应该记住的 5 件事
3	如果你甘于现状，并且已经在计划怎么用退休金了，那么你不是这个帖子的主角
...	...
n	作者：码农 时间：2016-01-01

由于已知从“title”标签中提取的标题大致是对的，但是由于有噪声词，不够精确，因此只需将“老程序员应该记住的 5 件事 _IT 新闻”与正文的每一行进行相似度比较，相似度采用最短编辑距离即可，只需计算出相似度最大的且达到规定阈值即可确定正文标题。此外，通过这种方法，还可以过滤掉正文中开始地方的嘈杂数据，以标题开始位置作为正文的正式开始位置，例如，本



例中即可过滤“IT 新闻 » 程序员”信息。

基于链接描述的方法和基于正文提取的方法各有优势，在实际中，多采用基于正文提取的方法，基于链接描述的方法准确度虽然高，但是会导致计算量非常复杂，一般用于验证标题提取准确率。也有通过“title”标签中字符串规则以分隔的方式进行提取的，例如，“—”“_”之前的文字可视为标题，这种方法对于多数网页是有效的，但是由于网页形式的多样化往往超出预期和想象，准确率不如基于正文的方式，甚至标题中因为本身带有“—”“_”进行分隔导致标题不完整。

6.3.5 网页的发布时间

获取网页的发布时间，有助于了解网页的时效性。可以通过元数据和正文内容进行网页发布时间确定。

(1) 基于 meta 标记获取。规范性网页大都会在 HTML 中通过 meta 信息标识网页发布时间，如下代码所示，表示网页的发布时间为 2015 年 12 月 09 日下午 15 时 53 分。

```
<meta name="og:time" content="2015年12月09日 15:53" />
```

(2) 基于正文内容抽取。网页发布时间在正文中的位置一般在首页标题下面或者在文章结尾处。通过正则表达式匹配正文的开始部分和结束部分也可以提取出时间。

基于元数据和正文内容方式，两者相辅相成，优先通过“meta”标记进行分析，倘若没有发现时间，再通过正文抽取获得相应发布时间。

6.3.6 网站的语言检测

网站的语言检测是搜索引擎认识网页的基础，只有在了解网页语言的基础上，才能使得后续的分析变得有意义。语言检测有如下两个目的：

(1) 爬虫深度确定。针对仅限于中文的搜索引擎，在检测出不是中文的网页之后，会调整爬虫爬行深度，对中文网页依然保持正常抓取深度，但对外文网页也许仅仅对一级域名和二级域名进行抓取，其余子页面则放弃。

(2) 语言分类限定。针对非语言限定的搜索引擎，例如 Google、Bing，它需要标识出网页的语言，进行分类归纳，便于全球各个地域都能优先使用本地语言搜索结果。“meta”中也有通过标识指定网页语言的，如下所示代码表示使用文本内容主要采用简体中文的方式。

```
<meta http-equiv="Content-Language" content="zh-CN">
```

当网页无法通过“meta”信息获得语言时，需要通过自然语言处理中的语种检测获得网页语言。

6.3.7 其他结构化数据

网页的 HTML 代码中还会定义一些其他信息，如 author、copyright 等，如表 6-9 所示，会根据实际情况，选择性获取这些结构化数据。在搜索结果页中，摘要旁边存在配图的情况，即通过元数据的“image”进行获得。

表 6-9 网页元数据中包含的其他结构化数据

meta 元数据	含 义	示 例
author	获取网页文档作者	<meta name="author" content=" 网易 " />
copyright	获取网页版权信息	<meta name="copyright" content="2011-2016 Iveely Inc." />
category	站长对网页的分类	<meta name="og:category " content=" 新闻资讯 " />
image	网页插图	<meta property="og:image" content="http://news.a.com/3fb95305d7b7146.jpg" /> <meta name="image" content="http://baike.bdiming.com/cms/static/baike.png">

作为搜索引擎的开发者，充分利用网页中的资源数据及元信息，可以给用户提供高质量的搜索体验和精准的搜索结果。

6.4 网页抓取策略

从互联网的角度来看，整个互联网即是一个巨大的网络。从数据结构角度来看，互联网是一个巨大的有向图，每个链接是整个有向图中的一个数据节点。理论上可以通过图的遍历方式实现爬虫对整个互联网的抓取，但是由于互联网的数据量远远超乎想象，数据质量也参差不齐，爬虫需要通过一定策略对互联网进行抓取，这些策略包括深度优先遍历、广度优先遍历和权重优先遍历。

(1) 深度优先遍历。是一种纵向搜索策略，是根据给定的种子链接，然后一个链接一个链接地深入下去，抓取到的链接集合将会被优先处理，直到处理完毕，再进行第二个种子链接，它是在搜索引擎早期的时候常用的遍历方式。图 6-12 所示为一个普通的网页链接关系图，根据图的优先遍历搜索，遍历路径为：A-F-I-E-H-D-J-B-C。

深度优先遍历容易使得爬虫数据向一边倾斜，且不易保证数据面的宽度，导致最终收录网页质量较差，但是在有的时候，深度优先效果比较好，比如垂直搜索中，具体使用方式还取决于使用场景。

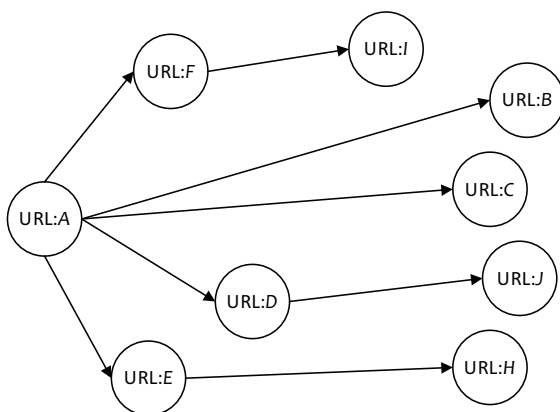


图 6-12 普通的网页链接关系图

(2) 广度优先遍历。是一种横向搜索策略，属于图的层次遍历，是通过每

次将新采集的链接放置到链接爬行队列的队尾，然后进行的遍历方式。根据图 6-12 所示的网页链接关系，遍历路径为：A-B-C-D-E-F-H-I。

(3) 权重优先遍历。广度遍历和深度遍历是常用的遍历方式，在工程中还会结合实际情况，采用更多的抓取策略，例如站点优先级遍历，根据站点的知名度选择。链接权重优先遍历，通过给链接打分的方式，选择分数高的链接优先进行抓取，但是都需要付出额外的代价。

综上三点所述，广度遍历适合全网搜索的爬虫，深度遍历适合垂直搜索和站内搜索的爬虫。

6.5 爬虫权限应对

目前众多网站出于种种原因，会实施反爬虫策略，如图 6-13 所示。

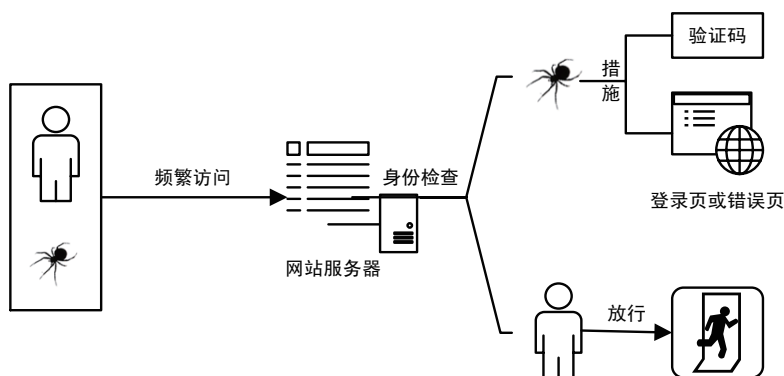


图 6-13 网站针对爬虫的反爬虫示意图

按照一般逻辑，网站是希望数据被更多搜索引擎收录，以使得更多网站流量流入站点，但是基于数据安全和网络拥塞两方面原因，实施反爬虫策略。

(1) 数据安全。不是所有的爬虫都是搜索引擎爬虫，也有部分爬虫为数据窃取者，它们对网站不存在任何利益价值。



(2) 网络拥塞。爬虫的数据抓取可能会造成宽带拥堵，爬虫越多可能性越大，通过购置高性能服务器会造成成本增加，因此，很多网站只允许主流搜索引擎爬虫对数据进行抓取。

爬虫的身份检查大多是基于“User Agent”进行验证，俗称用户代理，从浏览器角度来说，用户代理是一种浏览器标识，以 Chrome 浏览器为例，它的用户代理如下所示：

```
Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101 Safari/537.36
```

从爬虫角度来说，用户代理需要自行在访问的头部中加入用户代理相关信息，常用的搜索引擎均包含用户代理，例如百度网页搜索爬虫：

```
Mozilla/5.0 (compatible; Baiduspider/2.0; +http://www.baidu.com/search/spider.html)
```

因此，网站通过 UA 可以检测出是否来自浏览器的请求。此外，一些网站会对主流搜索引擎爬虫进行入白，即将相应的爬虫加入白名单，即使它频繁访问也允许，但是其他爬虫则不允许。

爬虫既然可以自行设定用户代理，在标准爬虫中，需要在用户代理中加入爬虫的名称，如百度网页搜索爬虫。

在爬虫进行数据抓取的情况下，反爬虫策略可能会导致爬虫产生各种异常情况，无论是在分布式环境还是在独立环境，异常情况主要针对爬虫的访问限制，这些限制包括禁止频繁访问、禁止未登录访问等。

(1) 禁止频繁访问。爬虫访问速度过快，会导致网站检测出异常流量，可能会根据 IP 等情况，对爬虫作出禁止访问（Access Denied）的情况，尤其是电商网站。

(2) Cookie 验证。Cookie 是网站用于辨别用户身份的常用方式之一，在访

访问网站服务器时，网站服务器有时会发送给访问端 Cookie，并保存在访问端，当下次访问端发起请求访问时，这时服务器会识别新请求的 Cookie，若识别成功，则允许访问，若失败则禁止访问。

(3) 验证码验证。验证码是一种比较有效的针对爬虫数据抓取的机制，在爬行过程中，增加验证码的输入，以确定不是爬虫正在获取信息。

(4) 登录验证。登录是网站对数据保护的一种增强方式。用户需要完成注册，然后登录其网站方可正常继续浏览网页。

需要解决上述问题，通过降低访问频率是不可取的，也有通过伪装 User Agent 的方式进行数据获取的。伪装成浏览器的 User Agent 或者知名搜索引擎的爬虫，都会被识别出来，例如，伪装百度网页搜索爬虫，将会被服务器通过 DNS 反解析的方式发现 IP 源并不属于百度。在拥有代理服务器的情况下，通过循环使用代理服务器解决问题，当前各类主流编程语言均提供代理访问方式。但是用户也需要了解禁止访问的真实原因，如果是没有严格遵守爬虫协议导致的问题，则需要修改爬虫规则。

爬虫在进行数据抓取的过程中，也要监控 Cookie 的设置情况，不仅仅是简单的数据获取，还需要考虑 Cookie 的设置，否则因为 Cookie 问题导致不能正常访问，有时需要对网站进行登录后才能访问，哪怕仅仅是游客身份，也会设置相应 Cookie。

验证码的难点在于不是所有的验证码都可以识别，简单的数字验证码可以通过文字识别技术识别，但是复杂的，例如需要通过计算、回答常识性问题、滑动鼠标等的验证码，当前则较难通过这种方式绕过。

在确定不是爬虫本身的问题时，可以通过 Cookie 伪装登录解决。通过手动登录，记录服务器 Cookie 的返回情况，将 Cookie 设置给爬虫，则爬虫在访



访问网站时带着 Cookie，很多访问权限也会被打开，为保证爬虫的顺利抓取，还需要关注 Cookie 的有效期，在有效期之后需要重新设置。

验证码和登录验证最终都会归于 Cookie 验证，但是总而言之，在爬虫被禁止访问网站资源时，第一需要考虑的是爬虫本身的爬行规则是否侵犯了网站本身，一般情况下，对于公开的数据，网站不会作出各种限制。

6.6 深网抓取

深网（Deep Web）又称为隐藏网络，是指互联网中不能被搜索引擎发现的非表现网络内容。与此相近的还有暗网（Darknet），暗网是采用非常规协议和端口进行连接的私有网络的集合，采用暗网的方式组织网站，目的在于分享一些敏感信息或者盗版电影，甚至发布非法信息等。暗网并不是不可访问，仅仅是服务器 IP 地址被隐藏而已。暗网属于深网的一部分，深网还包括一些搜索引擎无法发现的网站，但是原因与暗网不同，可能是搜索引擎没有找到发现其的网络路径。深网中包含了一些动态内容、私有网站及部分被限制访问的内容。

前面介绍过的网站地图也是获取网络中部分深网的方式之一，还有一种通过链接参数提交的方式获取，在现实中存在这样一个深网，网站内容都是动态生成，在没有触发的情况下，用户永远发现不了部分数据。爬虫则模拟用户不断请求的行为，不断通过请求动态页面生成的新链接，而当去抓取新链接之后，新链接页面又不断动态生成一些新链接，爬虫则可以通过不断地发现里面的数据，从而使隐藏的数据信息全部暴露出来。实质上搜索引擎本身也是一个深网，内部的很多数据无法直接展示，但是一旦有用户尝试搜索，信息将会展示，如图 6-14 所示，不断地发现关于“keyword”的链接，这部分展示的数据将会被

搜索引擎收录。爬虫试图通过输入参数值“keyword= 鱼”获得更多信息，类似操作大多发生在站内搜索中。

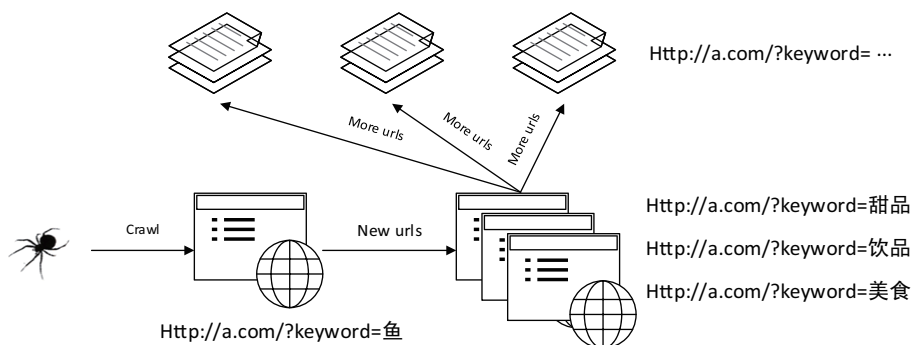


图 6-14 深网抓取示例

不同于其他类似的方式，例如，文章类网站常常以“http://b.com/?id=1”的方式通过参数请求从数据库获取数据并展示出来，与深网不同点在于，作为内容展示类网站，并不需要爬虫模拟输入参数，在网站首页有可能就存在该链接，深网通过爬虫自行输入参数获得信息。

6.7 抓取更新策略

网页在被抓取之后，会定期进行更新，发现更新页面后，将会重新抓取，传统方式采用基于 HTTP 协议头的内容分析，一般形式的 HTTP 协议头如下所示：

```
HTTP/1.0200OK
Date:Mon,31Dec201504:25:57GMT
Server:Apache/1.3.14(Unix)
Content-type:text/html
Last-modified:Tue,17Apr201506:46:28GMT
Etag:"a030f020ac7c01:1e9f"
Content-length:39725426
Content-range:bytes554554-40279979/40279980
```

通过验证 Last-modified 字段或者 Etag 标签进行验证数据是否存在更新。对

于目前互联网中的网页，大部分都是动态页面，甚至采用异步通信技术对网页更新，导致传统方式已经在当前无法满足验证网页更新需求，也有使用 MD 5 数字签名的方式，在服务器返回的网页文件流中，进行 MD 5 签名，将上一个 MD 5 签名和本地 MD 5 签名进行对比，若不一致则说明需要进行更新，值得说明的是虽然 MD 5 的签名速度非常快，但是数以亿计的网页都去做一次 MD 5 验证，还是一个工作量非常大的任务。

通过对网页更新频率的研究，网页的更新基本上符合泊松分布（Poisson Distribution）。泊松分布是一种结合统计学和概率论的数据分布理论，描述的是时间和事件发生频率的关系，对于网页更新即为网页发布之后的时间与其更新频率的关系，如图 6-15 所示。

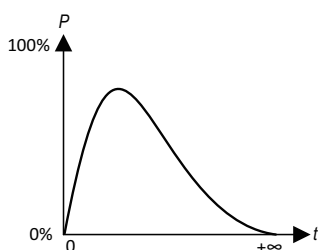


图 6-15 泊松分布示意

横坐标表示网页被抓取后的间隔时间，纵坐标表示在某时刻下，进行网页更新的网页比例，可以发现在网页被抓取后的最近时间更新频率最高，随着时间的累计，到后面更新的可能性非常低。因此，依据泊松分布对网页更新进行预测，然后再进行重新访问，达到较高访问效率，超过一定时间之后的网页，不再进行更新访问，泊松分布的访问间隔时间可以通过斐波那契数列进行，以 1 开始“1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144...”。对于给定网页，在爬取后间隔一天进行重访，若没有更新，则两天后再重访，依次类推，若这个过程中有更新则记录这个更新周期，下回从此时间间隔开始重访。

泊松分布对网页更新进度进行了预测，解决了大部分网页更新问题，但是仅仅依赖泊松分布不能解决所有问题，还需要其他辅助措施。例如，通过网页本身重要性进行定期更新，网页越重要则重访越频繁，还需要结合用户点击活跃度较高的网页进行优先级较高的重访。针对某一个页面，若一个页面不太重要，甚至不重要，对其迟一天访问和提前一天访问重要性不大。对于用户点击率较高的链接，优先级也会高于一般页面，进行重访的频率也应该更高。

6.8 本章小结

爬虫是整个搜索引擎中的重要功能之一。爬虫的架构系统直接关系到搜索引擎每天的数据采集量，采用分布式实时计算系统，不仅可以动态扩展爬虫工作节点，还可以使系统连续不间断地高效工作，不断地抓取网页，以及对网页进行结构化分析。爬虫中的抓取策略关系到搜索结果的数据质量，有效的数据总是以用户为中心，最大限度地满足用户的需求，数据的更新也是爬虫重要的任务之一，有效的数据更新策略可以使资源系统的使用最大化。采用符合泊松分布客观规律的抓取规则及集合斐波那契数列可以有效节省系统资源，使得爬虫有更多的时间去发现新数据。

第 7 章 大数据构建知识图谱

知识图谱是通过大数据提炼出的知识库，人类因为知识而伟大，搜索引擎将因为知识图谱变得更加聪明。因此，知识图谱是搜索引擎能够进行答案寻找、实时对话、信息预测的数据基础，它是一次从信息时代进入知识时代的革命性变化，知识图谱将会成为未来智能机器的“智慧之库，机器之心”。搜索引擎利用知识图谱作为智能搜索的入口，是技术发展的必然，正如 Google 的辛格博士在提到知识图谱时说的“The world is not made of strings, but is made of things”。知识图谱用图的结构描述着真实世界中的万千实体与实体关系。

7.1 概述

知识图谱是搜索引擎中非常重要的技术体系，它意味着搜索引擎技术已经到达了一个新的高地，当然这归结于这几年大数据的迅速发展。以笔者个人观点，知识图谱主要是为即将到来的人工智能搜索引擎提供数据基础和逻辑推理条件。当前各大互联网公司均有属于自己的百科网站，也是出于对下一代互联网的考虑。目前存在一些公开的知识图谱库，如表 7-1 所示。

表 7-1 目前公开的知识图谱库信息

知识图谱名称	内容信息来源	数量级别
Freebase	混合网络数据	6800 万实体，10 亿关系

续表

知识图谱名称	内容信息来源	数量级别
DBpedia	维基百科	1900 万实体，1 亿关系
Data.gov	美国政府官方网站	64 亿关系
wiki-links	维基百科	4000 万排除歧义的关系
Wolfram Alpha	计算知识	10 万亿实体

7.2 搜索引擎与知识图谱

在当前搜索引擎中，知识图谱通过知识关系拓扑结构，可以很好地解决用户精准问答，简单的知识图谱如图 7-1 所示。

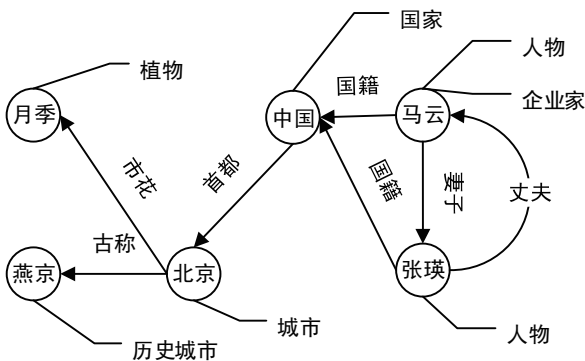


图 7-1 简单的知识图谱

图 7-1 中圆圈表示实体，箭头表示实体与实体之间的关系，其中“国籍”“人物”“企业家”等表示实体标签。

知识图谱对智能搜索的重要性在于可以快速获取搜索答案，而不再是一堆网页信息。

针对图 7-1，当用户搜索“马云妻子是谁？”，通过图谱直接定位“张瑛”，或者搜索“马云妻子国籍是？”，能够直接获得“中国”。第二个问题相对来说，

带有最简单的知识图谱的逻辑推理功能，首先定位到的是“马云妻子”，从知识图谱中给出答案“张瑛”，然后问题变为“张瑛国籍”，再寻找到最佳答案“中国”。

知识图谱不仅可以应用于常识性知识的问答，还可以应用于现实交通生活中，图 7-2 所示为简单的上海局部交通知识图谱。

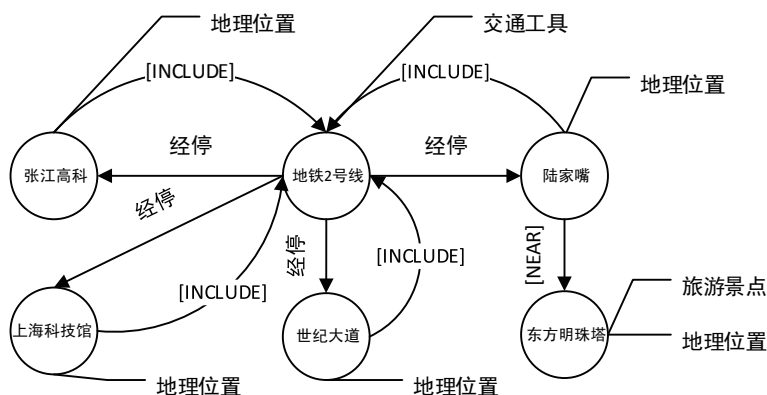


图 7-2 简单的上海局部交通知识图谱

针对图 7-2，当搜索“张江高科如何到陆家嘴”时，知识图谱则直接搜索“张江高科”与“陆家嘴”之间的交通工具，命中“地铁二号线”。同理，当搜索“张江高科怎么到东方明珠塔”时，知识图谱则搜索“张江高科”到“东方明珠塔”之间的交通工具，也会找到“地铁二号线”，并可以得到“东方明珠塔”距离“陆家嘴”地铁站也比较近，属于达到目的地的一种方式。图 7-2 中“[INCLUDE]”表示系统赋予的关系，非直接表述关系。类似的还有“[IS]”“[HAVE]”等。

综上所述，在当前的智能搜索时代，知识图谱至少在三方面提升了用户体验，如下所示：

(1) 直接命中答案。通过知识图谱直接找到用户最想要的信息，而不是通过一堆网页，让用户自己去慢慢寻找答案。

(2) 信息有条有序。知识图谱提供了最全面的信息参考,比如用户搜索“马云”,不仅可以看到马云的生平信息,还可以看到马云的财富、演讲等信息。

(3) 兼具到搜索的深度与广度。一个完整的知识体系,可以帮助用户在最短的时间内理解信息、提升效率,不必再换关键词去重新发起一次搜索。

7.3 可靠数据源选择

当前国内少有知识图谱构建的研究,一方面是基于成本,当前能够进行知识图谱研究的据笔者所知,微软亚洲研究院、百度公司及搜狗公司,他们都有足够的资金去做这样的事情;另一方面,目前已经有现成的 Freebase 等知识库,可以直接使用。但是无论怎样,正如大型互联网公司一样,很多核心数据都不希望依赖于外部。微软、百度公司及搜狗公司为在知识图谱上能够达到很好的效果,还利用各自的百科平台“必应网典”“百度百科”“搜狗百科”进行众包式的知识库补充。Freebase 的确是非常好的知识库,可遗憾的是,它对中文的支持并不多,而且存在重复情况,所谓重复情况是同一个实体,简体中文和繁体中文对其的描述被视为两条记录。

充分使用结构化的大数据,并且深入理解用户的搜索词,针对性地给出准确的答案,可以最大限度地满足用户对互联网信息的获取需求。

构建知识图谱的第一步是寻找最合适的数据源,有了数据源的准确性保障才能确定知识图谱的合理性。纵观国内外,维基百科是目前比较好的选择,维基百科由世界各地的志愿者合作编辑,其目标和宗旨是为全人类提供自由的百科全书。维基百科目前收录了近 500 万的英文条目,各类语言累计超过 2 200 万条目。无论从数据量的角度还是数据准确性的角度来讲都是不错的数据源,其他如百度百科、互动百科等也可以纳入到辅助数据之中。

7.4 实体抽取

实体抽取是抽取信息表达中的关系发起者和接受者。目前各大搜索引擎公司对于实体抽取标记会有不同。维基百科在对实体进行相关介绍时，遇到其他相关实体，则会通过链接的方式指向另外一个实体。因此，针对维基百科文章中的实体抽取时，可以通过爬虫分布式抓取相关网页获得在维基百科中尽可能多的实体。

维基百科正文中的链接指向另外的实体页面，因此，可以将该链接对应的词语作为一个实体。但不是所有正文中的实体都会被维基百科标记为一个链接，因此，要识别这些正文中的实体可以基于词性标注的方式。

实体的识别除在维基百科中获取已知实体信息之外，还可以通过自然语言处理中的分词技术与词性标注实现，根据词和词性识别实体信息。通过词性标注，确定的实体信息包括：人名、地名、机构团体名、其他专名、数词、数量词、时间。通过词性标注的方式进行实体抽取示例如表 7-2 所示，其中 nr 表示人名、ns 表示地理名、nt 表示机构团体名、qt 表示时间、nx 表示量词、nrf 表示翻译的外文人名。

表 7-2 通过词性标注的方式进行实体抽取示例

示例句子	实 体
马云 /nr 在 /p 杭州 /ns 居住 /v	马云（人名）、杭州（地名）
中国民主同盟 /nt 成立 /vi 于 /p 1941 年 /qt	中国民主同盟（机构团体名）、1941 年（时间）
五四运动 /nz 发起 /v 时间 /n 1919 年 5 月 4 日 /qt	五四运动（其他专名）、1919 年 5 月 4 日（时间）
比尔盖茨 /nrf 身高 /n 181cm/nx	比尔盖茨（人名）、181cm（量词）

在完成实体名称的抽取之后，还需要生成实体对应的标签（如“人物”“地理名词”等），这些词可以在分词的过程中通过词性进行获得，其次还可以通过本身数据源获得。例如，针对实体“北京”，维基百科自动为其建立分类，如图 7-3 所示。

分类: [中华人民共和国行政区划导航模板](#) | [中华人民共和国省级行政区](#) | [亚洲首都](#)
[中华人民共和国直辖市](#) | [北京](#) | [中国特大城市](#) | [国家历史文化名城](#) | [京津冀城市群](#)
[夏季奥林匹克运动会主办城市](#) | [亚洲夏季运动会主办城市](#) | [中国历代国都](#)
[国家中心城市](#) | [国家经济中心城市](#)

图 7-3 地理位置实体“北京”在维基百科中自动建立的分类

如图 7-3 所示, 针对实体“北京”在维基百科中自动建立的分类信息是对实体进行区分的特征信息, 这些特征信息将会是实体“北京”的实体标签, 标识它是独一无二的一个实体, 同样对于人物类的实体“汪涵”, 也存在图 7-4 所示的自动化分类信息。

分类: [1974年出生](#) | [在世人物](#) | [汪姓](#) | [苏州人](#) | [湖南卫视主持人](#)
[湖南大众传媒职业技术学院校友](#) | [湖南省政协委员](#)

图 7-4 人物类的实体“汪涵”在维基百科中自动建立的分类

这些分类信息将会直接转换为实体的标签, 将会对相同实体名区分、相同实体名合并产生重要影响, 它们是实体的唯一性特征标识。例如, 当用户搜索“湖南卫视主持人有哪些?”时, 将会通过实体的标签获得所有的湖南卫视主持人。同理, 也可以通过比较两个实体之间实体标签的交集数量去分析两个实体的关系紧密程度。

在工程应用中, 和维基百科一样, 会对实体标签建立相应的索引和缓存, 使得用户的搜索能够尽快定位到相关实体当中。

7.5 关系抽取

实体关系抽取的方法在理论上有很多种, 包括基于模式匹配的方式、基于机器学习的机制、基于 Ontology 的关系 (利用哲学中的存在论关系) 抽取方式、基于词典驱动的关系抽取方式及基于混合的抽取方式, 都属于典型的关系抽取技术。



7.5.1 关系抽取概述

在早期实体关系抽取研究领域常常使用基于模式匹配的方式，通过指定模式关系列表，首先将文本和模式匹配，匹配成功后，根据模式中的关系约定直接提取。这种方式处理英文，可以得到很好的效果，但对于丰富灵活的中文，虽然它提取准确率较高，但覆盖率却比较低，且过于依赖模式，不够灵活，在少量数据集上（如实验室测试）可以采取此种方法，可以得到较好的结果，对于搜索引擎数亿级的海量数据抽取，不太合适。

其他方式也各有所长，但是在工程领域，尤其是在海量数据的分析上，最常使用的是基于机器学习的方式，海量数据便于发现数据特征，利用特征的方式去解决问题。因此，基于机器学习的方式进行关系抽取，实质是将关系抽取问题视为分类问题，通过机器学习算法，首先在人工标注的语料库上进行特征学习，然后将学习到的特征应用到预测数据中，最终得到期望结果。

在工程应用中，实体关系的抽取包含隐藏关系抽取和确定关系抽取。

（1）隐藏关系抽取。语句中没有明显标识的关系词，例如“中国上海很漂亮”，其中“中国”和“上海”之间隐藏了包含关系（[INCLUDE]），“重庆在长江上游”，其中“重庆”和“长江上游”形成位置关系（[LOCATION]），“阿里巴巴公司的张云表示情况很乐观”，其中“阿里巴巴”和“张云”形成雇佣关系（[EMPLOY]）等。隐藏关系是中文语言表达方式中常常使用的方式，此类隐藏关系大多为常识性知识信息。

（2）确定关系抽取。语句中已经明确实体间的关系，例如“阿里巴巴董事会主席马云开始演讲”，这里已经明确“阿里巴巴”与“马云”的关系，马云是阿里巴巴的董事会主席，董事会主席已经是确定关系。确定关系抽取是将互联网文档中已经确切说明和描述的实体关系抽取，这类关系抽取的特征主要取

决于语言的不同表达方式，即使在中文和日文之间的抽取方式也存在差异。

7.5.2 隐藏关系抽取

解决隐藏关系的抽取可以采用支持向量机（Support Vector Machine, SVM）方式。众所周知，支持向量机擅长解决分类问题，因此，对于隐藏关系的抽取，支持向量机实质上是通过 S 将隐藏的关系类型进行分类，确定其属于哪种隐藏关系。隐藏的关系通过预先定义，总体包括：包含关系（[INCLUDE]）、位置关系（[LOCATION]）、雇佣关系（[EMPLOY]）等。

在利用支持向量机进行实体隐藏关系抽取之前，需要深入了解支持向量机的内部机制。支持向量机通过构建 N 维（ $N>1$ ）的超平面对数据进行分类，这个超平面即是分类边界。

图 7-5 所示为线性分类的例子，斜线即超平面。一般情况下，平面内的点距离超平面的距离可以作为分类预测的准确程度评判依据。若某一超平面是支持向量机的最大间隔值，则这个平面也称为最大间隔超平面，此时，这个支持向量机的分类器又称为最大间隔分类器。

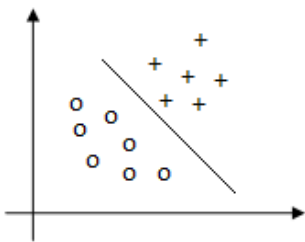


图 7-5 支持向量机的简单超平面示例

从分类效果上，图 7-6 所示的三个超平面中，中间粗体超平面达到的分类效果最佳，图 7-5 和图 7-6 介绍的是线性可分的例子，可以在一个平面中表示其特征，但是一般要求解的问题，常常是线性不可分的，如图 7-7 所示。

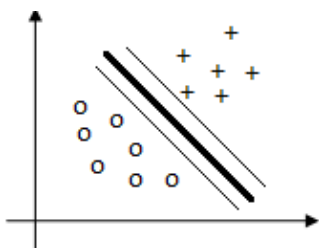


图 7-6 支持向量机最大间隔超平面示例

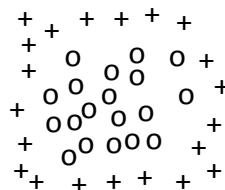


图 7-7 支持向量机的简单线性不可分示例

此时，常用做法是将数据特征映射到高维空间，如图 7-8 所示。

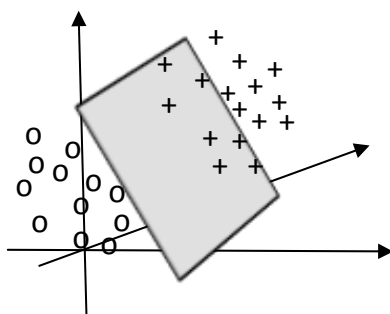


图 7-8 支持向量机的高维空间示例

线性不可分的数据映射到高维空间时会带来一定的问题，维度越高计算程序越复杂。因此就有了核函数的产生，它的价值在于核函数将特征进行低维转换到高维，但是它依然在低维进行计算。即使表现形式展现为多维，但避免了在高维上直接进行复杂的数学计算，因此支持向量机对于非线性分类有很好的优势。

实体关系抽取采用支持向量机，核心在于选择合适的特征集。

(1) 实体词特征集。一个实体的名称肯定是由一个词或者多个词组合而成的，在一个具有关系的实体句子中，至少包含两个实体。

其中，实体集记作 E ：

$$E = \{E_1, E_2, \dots, E_n\}$$

每个实体名称 E_i 的词集特征记作 W_i ：

$$W_i = \{W_{i,1}, W_{i,2}, \dots, W_{i,m}\}$$

$W_{i,j}$ 表示组成实体 E_i 的第 j 个词。

(2) 实体词性特征集。组成实体名称的一个词或者多个词标注的词性特征。每个实体名称 E_i 的词性特征记作 T_i 。

$$T_i = \{t_{i,1}, t_{i,2}, \dots, t_{i,k}\}$$

(3) 实体标签特征集。每个实体 E_i 都有属于自己的属性，例如“人物”“地理位置”“省市名称”等，记作 A_i 。

$$A_i = \{a_{i,1}, a_{i,2}, \dots, a_{i,f}\}$$

(4) 实体距离特征集。在一个具备实体关系的句子中，两个存在关系的实体在句子中的位置差特征，记作 D 。其中， $d_{i,j}$ 表示第 i 个实体距离第 j 个实体的距离，即中间字符串长度。

以句子“张学友出生于 1961 年 7 月 10 日”，获取上面各项特征集，先分词并进行词性标注后为“张学友/nr 出生于/v 1961 年 7 月 10 日/qt”，进行特征抽取，如表 7-3 所示。

表 7-3 特征抽取示例

特征集	结 果
实体特征集 E	张学友、1961 年 7 月 10 日
实体组成词集 E_i	$E_1 = \{\text{张学友}\}$, $E_2 = \{1961 \text{ 年 } 7 \text{ 月 } 10 \text{ 日}\}$
实体词性特征集 T_i	$T_1 = \{\text{nr}\}$, $T_2 = \{\text{qt}\}$
实体标签特征集 A_i	$A_1 = \{\text{人物、香港歌手}\}$, $A_2 = \{\text{时间日期}\}$
实体距离特征 D	$d_{1,1} = 3$

表 7-3 中的信息均是通过语料库中训练学习得到的，语料库的选择非常重要，语料库中不仅需要分词且需要进行词性标注、实体标注、实体标签标注及实体隐藏关系标注，标记出实体之间的隐藏关系是使用支持向量机进行分类的

基础。整个过程中，抽取的特征类型越多，计算的时间开销越大。语料库准备好之后，方能进行下面的工作。

(1) 特征抽取及特征向量的构造。

(2) 对特征向量降维处理。利用特征选择算法对第一步的特征向量进行特征选择，重新构造新的低维特征向量。

(3) 构造 SVM 分类器。

支持向量机分类器构造完成之后，即可完成实体关系抽取。

7.5.3 结构化确定关系抽取

所谓结构关系抽取，是指可直接获取的实体及其关系。以维基百科为例，其中已经存在一些人工标注好的实体关系，例如，访问“<https://zh.wikipedia.org/zh-cn/>上海市”将会看到如图 7-9 所示的结构化信息。

行政区类型	直辖市
政府所在地	黄浦区
最大区县	崇明县
市委书记	韩正
人大常委会主任	殷一璀
市长	杨雄
政协主席	吴志明

图 7-9 维基百科关于上海市的部分结构化数据

图 7-9 中的信息可以直接通过 HTML 分析获取，而不需要通过机器学习的方式获得。将图 7-9 中的信息转变为知识图谱，如图 7-10 所示。

补全图 7-9 中的数据，如表 7-4 所示，根据表 7-4 即可绘制出知识图谱图形化的表示方式图 7-10。同理，实体识别采用分词和词性标注识别，如表 7-5 所示。

表 7-4 维基百科中关于“上海市”实体关系的补充

关系发出实体	指向关系	关系接收实体
上海市	行政区类型	直辖市
上海市	政府所在地	黄浦区
上海市	最大区县	崇明县
上海市	市委书记	韩正
上海市	人大常委会主任	殷一璀
上海市	市长	杨雄
上海市	政协主席	吴志明

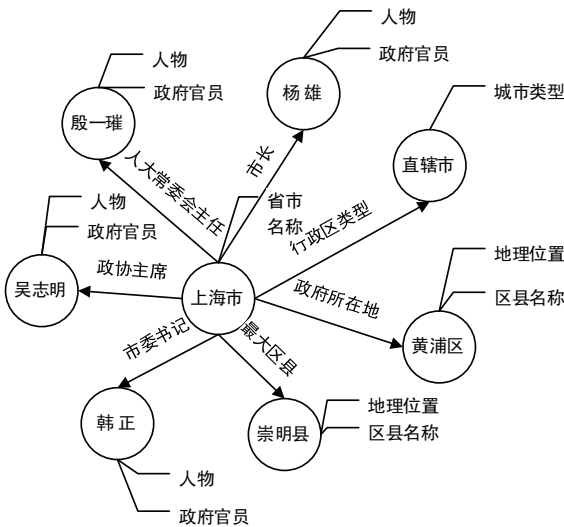


图 7-10 根据维基百科关于上海市局部数据构建的知识图谱

表 7-5 维基百科中关于“上海市”实体关系的分词处理

关系发出实体	指向关系	关系接收实体
上海市 /ns	行政区 /n 类型 /n	直辖市 /n
上海市 /ns	政府 /nis 所在地 /n	黄浦区 /ns
上海市 /ns	最大 /gm 区县 /n	崇明县 /ns
上海市 /ns	市委 /nis 书记 /nnt	韩正 /nr
上海市 /ns	人大常委会 /nis 主任 /nnt	殷一璀 /nr
上海市 /ns	市长 /nnt	杨雄 /nr
上海市 /ns	政协 /nis 主席 /nnt	吴志明 /nr

此类结构化的确定关系抽取相对比较简单，结构化数据实质上是人工已经参与的数据。这部分数据的显著特征为短语居多、关系明确、基本无歧义，属于简单的实体关系集。复杂的关系集合需要通过非结构化文本抽取。

7.5.4 非结构化确定关系抽取

7.5.3 中通过维基百科中已经标记的结构化实体确定关系进行抽取分析。但在实际的维基百科中，大部分数据是非结构化的，需要进行自动化抽取分析，例如，维基百科中关于“上海市”页面的第一句话：

上海，简称沪，别称申，中华人民共和国直辖市、国家中心城市之一，也是中国最大的城市。

工程中，期望能够将上面一句话进行实体和实体关系抽取，提取出如图 7-11 所示的知识图谱。

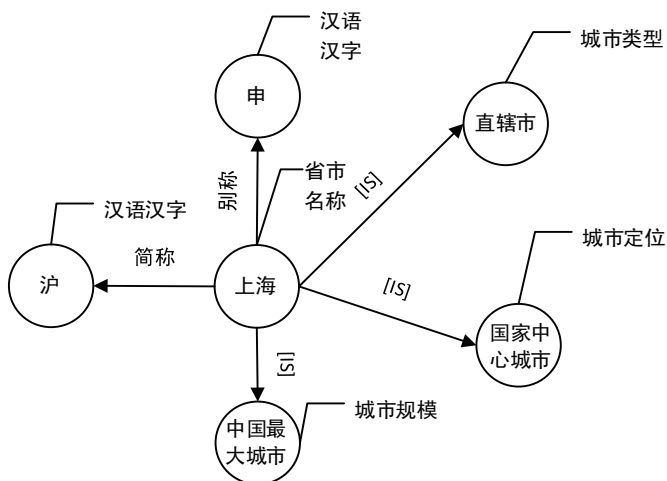


图 7-11 非结构化确定关系抽取简单示例

对于确定关系的抽取，需要从语言学的角度及自然语言的语句关系入手。笔者曾经试图通过语句关系结构来解决此问题，例如，利用斯坦福大学的自然

语言处理解析文本，对“云南位于中国西南的边陲”进行分析示例，如图 7-12 所示。

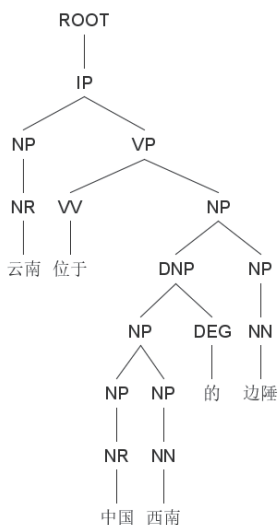


图 7-12 斯坦福大学自然语言处理语句关系示例

由于中文句式多变，句式嵌套的复杂句型较多，因此试图通过语句关系获得特征，目前可行性依然较低。但是最终笔者依然探索到一种新的方式，基于词性序列的句子分析，简单地说是通过词性的相关性序列进行抽取，同基于统计的分词方法类似，只不过观察序列不再是字而是词性。

语料库的选择非常重要，语料库的构建过程也是非常复杂的过程，尤其是在无现成语料库的情况下。

(1) 原始语料库预处理。对原始词性语料库进行系统标注实体、标注实体标签特征。

(2) 人工标记实体中的关系信息。不同于已经约定好的关系集合，例如“雇佣关系”“包含关系”“位置关系”等，实体之间的关系大多数都是在文本中自动发现的，需要人工在语料库中标记出关系词组的位置。

(3) 形成最终语料库。部分语料库形式如表 7-6 所示。

表 7-6 非结构化确定关系抽取语料库示例

句子编号	句 子
1	张学友 nr/ea 出生于 v/r 1961 年 7 月 10 日 qt/eb
2	重庆火锅 nf/ea, 也 d/n, 称作 v/r, 毛肚火锅 nf/eb, 或 c/n, 麻辣火锅 nf/eb
3	黄河 ns/ea, 发源 vi/r, 于 p/n, 玛曲 ns/eb
4	...

表 7-6 中的“张学友 |nr/ea”表示“张学友”的词性为“nr”是实体关系中的“实体 a(entity a)”，“1961 年 7 月 10 日 |qt/eb”表示“1961 年 7 月 10 日”的词性为“qt”是实体关系中的“实体 b (entity b)”，“出生于 |v/r”表示“出生于”的词性为“v”是实体关系中的“关系 (relation)”，“也 |d/n”表示“也”的词性为“d”，不属于实体关系中的实体和关系两部分，“n”是“null”的缩写。状态“ea”和状态“eb”必须成对出现。

(4) 设计算法模型。在给定算法模型之前，需要思考一个问题，给定一个已经分词并标注好词性的语句，例如“李彦宏 |nr, 毕业 |v, 于 |p, 北京大学 |nt”，实质是寻找“李彦宏”在词性为“nr”时，是实体的概率，以及“毕业”“于”是关系的概率，对于词性“nr”“nf”等本身就可以视为实体，因此，主要还是在实体已知的情况下，“毕业”“于”或者“毕业于”是关系的概率问题，既然是概率问题，需要通过概率统计学习的方式，因此问题已经转换为判别式概率计算问题。介于语料库中存在观察序列、存在各自是否是“ea”“eb”“r”“n”的状态可能性，因此利用条件随机场（CRF）即可求解。介于马尔科夫链的假设是每个状态只与它前面的状态有关，这样的假设对于实体关系抽取显然是有偏差的，一个词或者多个词是否是实体关系与其词性有关，而词性是通过汉语语言全局表达的方式确定的，因此，选择条件随机场更有利于计算。

条件随机场模型是 Lafferty 于 2001 年在最大熵模型（MEM）和隐马尔

科夫模型（HMM）基础上提出的一个判别式概率无向图学习模型，广泛用于标注和切分有序数据的条件概率模型，关于条件随机场的详细介绍可参考 3.1 节。

基于表 7-6 所示的语料库，确定模型各项参数：

（1）观察序列。所有的词性，语料库中词性所属词语不作为观察序列，中文的实体关系与具体词无关。所有观察序列包括：nr1、nrj、ns、nt、nl、nr2、nrf、nsf、nz、ng、n、t 等。

（2）序列状态。“ea”“eb”“r”“n”每个词性只有这四种状态。

（3）训练模型。第一步，计算每个词性出现的次数；第二步，计算每个词性状态为“ea”“eb”“r”“n”的概率；第三步，计算每个词性转移到下一个词性的状态概率，例如，当前词性“nr1”的状态为“ea”，则转移到下一个词性状态为“eb”的概率，每个词性的计算结果均为一个 4×4 矩阵；第四步，计算上下文关系概率，这也是条件随机场区别于隐马尔科夫模型的地方。计算每个词性在四种状态下相邻下一个词性的概率及相邻上一个词性的概率。

（4）执行实体关系抽取。第一步，进行分词词性标注，利用分词系统和词性标注系统对目标语句进行相应操作，例如“英国|ns, 首都|n, 位于|v, 伦敦|ns”；第二步，提取词性序列，对于示例“英国首都位于伦敦”提取出序列“ns n v ns”；第三步，通过维特比算法进行最可能状态标注，对序列“ns n v ns”标注后得出“ns/ea n/r v/n ns/eb”，意味着“英国|ns/ea, 首都|n/r, 位于|v/n, 伦敦|ns/eb”，因此得出实体为：英国、伦敦，关系为：首都；第四步，标记实体标签，词“英国”的属性为“国家”，词“伦敦”的属性为“地理位置、省市名称”。

（5）最终得出知识图谱，如图 7-13 所示。

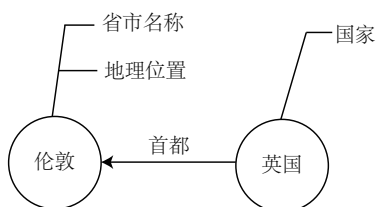


图 7-13 “英国首都位于伦敦”知识图谱

对于一个完整的句子直接按照上述方法进行操作，最终通过维特比解码可以解决问题。但实际工程实践却不容易直接使用，原因在于互联网上各式各样复合的句子、短语句，提取到的效果并不明显，需要对目标句子进行预处理，目的是使目标句子转变为独立完整的句子。

(1) 代词转换。理所当然代词不具备实体标签，但它指向的信息可能是一个实体，因此需要将代词替换为代指的词语。例如，对于句子“朱自清，原名自华，号秋实，后改名自清，字佩弦，原籍浙江绍兴，他出生于江苏省东海县”，此时此刻需要变更为“朱自清，原名自华，号秋实，后改名自清，字佩弦，原籍浙江绍兴，朱自清出生于江苏省东海县”。

(2) 实体补充。对于语言表达的一句话中，表述核心为同一对象，需要对不完整的语句进行对象补充，便于进行实体关系分析。例如，将“朱自清，原名自华，号秋实，后改名自清，字佩弦，原籍浙江绍兴，朱自清出生于江苏省东海县”补充为“朱自清，朱自清原名自华，朱自清号秋实，朱自清后改名自清，朱自清字佩弦，朱自清原籍浙江绍兴，朱自清出生于江苏省东海县”。

(3) 句子拆分。每个句子都具备了整体描述信息，将句子拆分后，进行编码将会得到更好的效果。例如，将“朱自清，朱自清原名自华，朱自清号秋实，朱自清后改名自清，朱自清字佩弦，朱自清原籍浙江绍兴，朱自清出生于江苏省东海县”拆分为表 7-7 所示的内容。

表 7-7 实体补充之后的句子拆分示例

句子编号	句 子
1	朱自清原名自华
2	朱自清号秋实
3	朱自清后改名为清
4	朱自清字佩弦
5	朱自清原籍浙江绍兴
6	朱自清出生于江苏省东海县

分别解码。对拆分后的每句话进行解码，抽取实体关系信息，最终确定的实体关系如表 7-8 所示。

表 7-8 非结构化实体关系抽取示例

关系发出实体	指向关系	关系接收实体
朱自清	原名	自华
朱自清	号	秋实
朱自清	字	佩弦
朱自清	原籍	浙江绍兴
朱自清	出生于	江苏省东海县

7.6 知识图谱检测

在通过实体抽取及实体关系抽取之后，知识图谱的基本模型已经构建完毕，但是要达到完整、可靠，还需要对知识图谱中各个实体及实体关系进行进一步处理，以保障知识图谱中的数据能够为用户带来价值，而不在某些有歧义甚至错误的地方误导用户。知识图谱的检测主要包括实体关系修正、实体对齐及实体歧义分析。

7.6.1 实体关系修正

在完成实体关系抽取之后，获得的数据信息并不一定完全可靠。需要通过

不断地在互联网信息中抽取，并对每一个关系进行投票，例如，对一篇文档进行抽取后，得到如图 7-14 所示的实体关系。

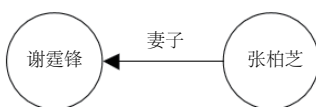


图 7-14 初步分析获得的实体“谢霆锋”与“张柏芝”的关系

图 7-14 中，实体“张柏芝”被识别为实体“谢霆锋”的妻子，但是目前两者并不存在夫妻关系，有可能是由于分析了过时的文档导致的。因此需要通过投票的形式去纠正实体关系。通过分析当前更多关于实体“谢霆锋”与“张柏芝”的关系，发现绝大多数认为实体“张柏芝”是实体“谢霆锋”的前妻，而并不是妻子，所以通过修正后的实体关系如图 7-15 所示。

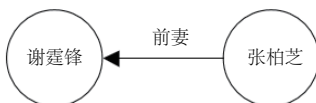


图 7-15 对实体“谢霆锋”和“张柏芝”进行实体关系修正的结果

7.6.2 实体对齐整合

在进行实体抽取过程中，由于实体的各个属性与关系并不一定来自同一个文档。换个角度理解，即实体具有不同的 ID，但是在现实中却代表着同一对象的实体。例如，针对知识图谱中的实体“朱自清”和实体“朱自华”，实质它们都是指向同一实体，只是称呼不同罢了（朱自华是朱自清的原名），因此需要将这样的实体进行合并，整个实体合并的过程称为实体对齐。

通过不断研究和发现，进行实体对齐的相关实体信息遵守如下几个规则：

- (1) 具有相同属性与属性值的实体。
- (2) 实体对外实体关系中，绝大部分实体关系存在子集或者包含关系。

(3) 实体字符描述语义相似度极高。

继续以“朱自清”和“朱自华”两个实体为例，如图 7-16 所示。

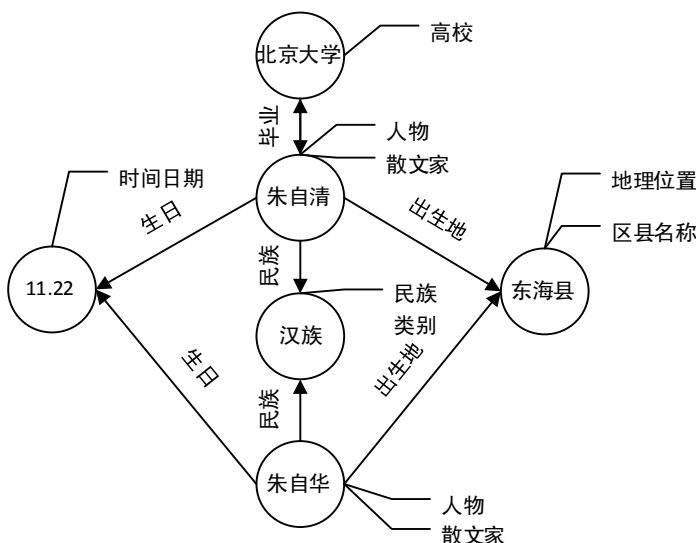


图 7-16 实体“朱自清”和“朱自华”在知识图谱中的关系

通过观察发现，“朱自清”和“朱自华”的属性一致。“朱自华”的对外实体是“朱自清”的对外实体的子集，两者语义相似度 0.98。因此综合考虑，确定其描述的是同一实体，两者可视为同义词。

实体对齐的好处在于，在实体对齐之前，用户搜索“朱自华毕业于哪所高校？”，按照已有的知识图谱是无法给出具体答案的。但是二者语义对齐之后，当系统检索“朱自华毕业高校”无果之后，会自动转换为“朱自清毕业高校”进行检索，并给出答案“北京大学”。通过实体对齐，可以让内部知识更加丰富，搜索过程更加灵活。

但是在上亿实体中，并不是将所有实体进行两两比较属性、对外关系实体，而是先通过聚类的方式，让可能相似的放在一个聚簇空间之后，然后通过相同的属性索引，进行实体比对。例如，对拥有一亿实体信息知识图谱，若两两

比较，即使有再多的云计算服务器，也很难计算完毕，但是通过实体标签索引，例如“人物”，则将所有属性为“人物”的实体取出进行比较，再通过云计算平台进行实体对齐就变得比较简单。

7.6.3 实体歧义分析

不同实体 ID 有描述同一实体的可能性，在实体对齐过程中，可能存在这样的情况：相同实体名称却表示的不是同一个实体。实体名“李湘”的知识图谱信息如图 7-17 所示。

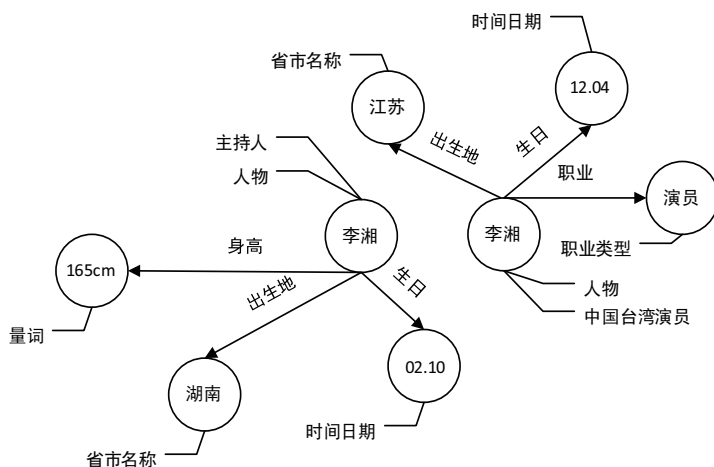


图 7-17 知识图谱中的不同实体名“李湘”

通过图 7-17 可以发现，存在两个名为“李湘”的实体，倘若这个时候，用户搜索“李湘生日是什么时候？”，实体的歧义就出现，系统可以定位到两个实体，需要集合实体标签给出相似实体的差异并给出结果“主持人李湘生日是 02.10，相似人物中国台湾演员李湘生日是 12.04”。

在属性不同的时候，系统很容易消除这种歧义。假设还有另外一个相同实体名为“李湘”的人，属性也为“主持人”“人物”，这个时候，希望从属性中消除歧义就比较困难，但是只要没有达到实体对齐的原则，那么无论是在实体

标签或者实体对外关系上一定是有差异的，在检索过程中将这种对外差异性体现出来也可以帮助系统消除歧义。但是也存在这样的情况，名称为“李湘”的实体数量不止两个，有成百上千个可能，用户在搜索关键词“李湘”时，如何确定哪个实体返回到搜索页面（一般显示在搜索引擎的知识卡片中）。这个筛选过程需要通过计算实体权重（EntityRank）完成。

在 3.7 节中，有对关键词权重（TextRank）的介绍，实体权重的原理和关键词权重类似，以及与知名的网页排序算法 PageRank 也非常类似。从数据结构角度来讲，知识图谱实际上是一个图结构，图中的实体之间可能存在关系，将这种关系视为投票，一个实体的“投票数”由指向它的其他实体决定，指向它的实体，相当于给它投一票。在初始状态，每个实体的权重值都一样，通过不断的迭代计算“投票数”，最终计算出每个实体的权重。实际上，计算出每个实体权重之后，权重越大的实体则它们的重要性越高，在搜索过程中，优先展示。

7.7 知识推理与计算

知识图谱中虽然有上亿的实体，但仅仅依靠这些实体去完成用户的问答是比较困难的，对于比较灵活的问题，往往不能返回用户满意的结果，因此知识推理的产生用于发现隐含的知识信息，它只是用于计算实体间的差异和关系度。

7.7.1 知识推理

知识推理主要是针对实体对外关系做的逻辑推理，图 7-14 所示为知识图谱的简单逻辑推理。

当用户搜索“李湘女儿多大了？”，则首先通过“李湘女儿”推导出“王诗龄”，再推导出“6 岁”，并给出精准答案。

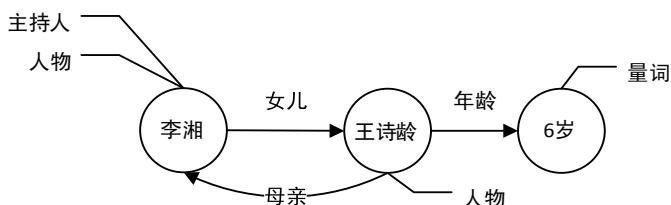


图 7-18 知识图谱的简单逻辑推理

知识推理的核心是实体之间关系的相互推导，关系推导过程中最重要的是关系识别与匹配。例如，上述例子中用户搜索“李湘女儿多大了？”，实体关系中恰好存在“女儿”这一关系，但有时候并不都是完全匹配的，同样是针对图 7-18 中的知识图谱信息，如果用户搜索“李湘孩子多大了？”，则需要考虑问题的复杂性，因为孩子包括“儿子”和“女儿”，需要将实体“李湘”对应关系中的“儿子”和“女儿”均提取出。在工程应用中一般不会将“孩子”拆分为“儿子”和“女儿”，而是计算“孩子”和“女儿”的语义相似度，超过一定语义相似度的均可提取出，语义相似度计算可参考 3.4 节内容。

7.7.2 知识计算

知识计算是实体与实体之间相互关系的动态结果，目前知识计算主要运用于知识信息值计算与知识关系度计算。

(1) 知识信息值计算。是对知识推理的多次进行，如图 7-19 所示。

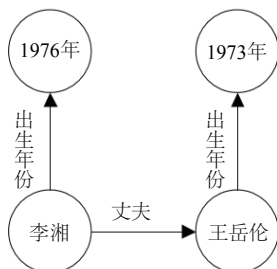


图 7-19 实体“李湘”与“王岳伦”的出生年代

已知实体主持人“李湘”生于 1976 年,实体导演“王岳伦”出生于 1973 年,当用户搜索“王岳伦比李湘大多少岁?”时,通过知识图谱查找将两者生日年份进行计算,并返回给用户精准结果“3 岁”。

知识信息值不仅可以计算年份时间,还可以计算身高、海拔、重量等,只要与数值相关均可。知识信息值计算是实体之间相互比较的基础,例如,当用户搜索“购买 Surface 4 好还是购买 Macbook 好?”时,可以通过将两实体之间的信息值进行计算,体现两者在各个属性之间的差异。

(2) 知识关系度计算。知识的关系度是指知识信息的关联紧密程度,在知识图谱中可以很好地计算,原则上是两个实体在知识图谱中的距离越远,关系度越低。要进行关系度计算,可以计算知识图谱中两个实体的最短路径。

计算两个实体的最短路径,可以通过 Dijkstra 算法实现,实体信息如图 7-20 所示。

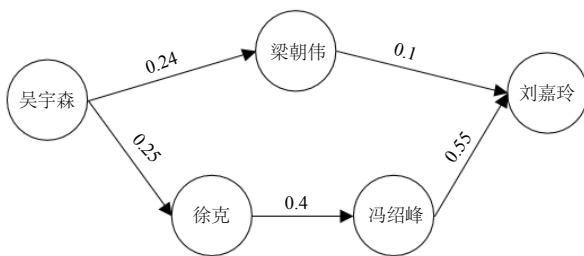


图 7-20 实体中的关系图结构示例

根据图 7-20 所示信息,实体与实体之间的数值表示实体之间的疏密度,值越小表示关系越紧密,通过 Dijkstra 算法计算实体“吴宇森”与实体“刘嘉玲”的疏密度。Dijkstra 算法又称位单源最短路径,所谓单源是在一个有向图中,从一个顶点出发,求该顶点至所有可到达顶点的最短路径问题。

Dijkstra 算法是运用贪心的策略,从图分析的起始位置开始,不断地通过相关联的点找出到其他点的最短距离。它的基本思想是:设置一个顶点的集合 S ,

并不断地扩充这个集合，一个顶点属于集合 S 当且仅当从源点到该点的路径已求出。开始时 S 中仅有起始位置点，并且调整非 S 中的点的最短路径长度，找当前最短路径点 U ， U 表示图中的一个顶点，将其加入到集合 S ，直到终点在 S 中。针对图 7-15，Dijkstra 算法的迭代过程与集合 S 的变化如表 7-9 所示。

表 7-9 Dijkstra 算法的迭代过程与集合 S 的变化

迭代次数	S	U	“梁朝伟”	“徐克”	“冯绍峰”	“刘嘉玲”
初始	{“吴宇森”}	—	0.24	0.25	MAX	MAX
1	{“吴宇森”，“梁朝伟”}	“梁朝伟”	0.24	0.25	MAX	0.34
2	{“吴宇森”，“梁朝伟”，“刘嘉玲”}	“刘嘉玲”	0.24	0.25	0.65	0.34
3	{“吴宇森”，“徐克”，“冯绍峰”，“刘嘉玲”}	“刘嘉玲”	0.24	0.25	0.65	0.99

根据表 7-9，“吴宇森”与“刘嘉玲”的最短路径如图 7-21 中的虚线所示。

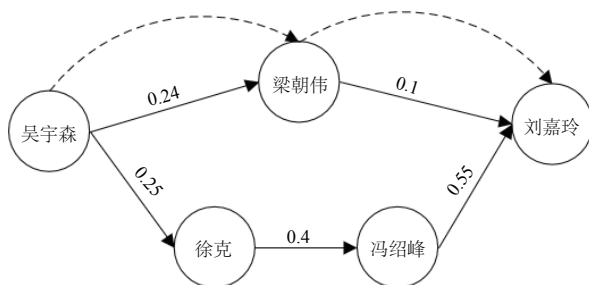


图 7-21 实体“吴宇森”与“刘嘉玲”关系的最短路径示例

知识关系度计算有助于分析实体之间的关系，例如，用户在搜索“吴宇森与刘嘉玲关系”时，可以通过最短路径进行描述为“刘嘉玲是吴宇森好朋友梁朝伟的妻子”，如果没有寻找到最短路径，则可能被描述为“刘嘉玲是吴宇森好朋友徐克的合作演员冯绍峰的朋友”，显然通过最短路径的描述最简洁也最合理。知识关系度计算在搜索过程中不仅可以为用户提供相关性推荐，还可以从信息的深度方面分析某个实体更深、更广的综合信息。

7.8 知识聚类

知识聚类的目的是体现知识在现实互联网中的关联程度，将关联程度较为紧密地归集在一起。

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) 是一种基于密度的聚类算法，在 DBSCAN 算法中，预先定义两个变量，一个是 ϵ 表示半径范围，另一个是 MinPts 表示半径内指定的点的数量，还包括直接密度可达、密度可达和密度相连三个概念。

(1) 直接密度可达。对于样本集合 N ，给定样本点 q ，如果 q 在核心点 p 的半径范围内，则称为数据点 q 与核心点 p 直接密度可达。

(2) 密度可达。给定样本集合 N 中对象链 $P_1, P_2, P_3, \dots, P_n$ ，如果 $P_1=q$ ， $P_n=p$ ，如果 P_{i+1} 在核心点 P_i 的半径范围内，则 p 与 q 密度可达。

(3) 密度相连。数据集合中，如果数据点 A 与其中的数据点 P 和数据点 Q 都是密度可达，则认为 P 和 Q 密度相连。

DBSCAN 算法的思想是：在以某点为核心点的基础上，在指定半径范围内，拥有超过指定的点数量，则形成一个聚簇。DBSCAN 的点被分为核心点、边界点和噪声点。

(1) 核心点。在该点的指定半径范围 (ϵ) 内有超过指定数量 (MinPts) 的点，属于密度稠密区内部的点。

(2) 边界点。在指定半径范围内且附近数量小于 MinPts，但是却落在某核心点的领域内，属于密度稠密区边缘上的点。

(3) 噪声点。不在核心点的领域内，且在指定半径范围内也不存在达到 MinPts 数量的点，即核心点和边界点之外的点。

DBSCAN 计算流程：首先将所有需要聚类的数据标记为未访问；其次随机选择一个数据对象 p ，将 p 标记为已经访问，如果 p 的半径范围之内存在 MinPts 个对象，则认为 p 是核心点，相应创建一个聚类簇 C ，将 p 添加入 C ；然后将 p 半径范围内所有的对象设置为 N ，对其中的每个点进行遍历，如果是未访问的数据点，则将该数据点标记为已访问，如果该数据点在半径范围内存在的数据对象不少于 MinPts 个，且该数据点还不属于任何聚类簇，则可将此数据点视为以 p 为核心点的聚类簇 C 的成员；依次遍历，直到最后所有的点被标记为已访问。在访问过程中，如果 p 不满足在半径范围内数据对象不少于 MinPts 个，则认为 p 为噪声点。

Kmeans 和 DBSCAN 算法都是常见的聚类算法，但是二者的区别在于 DBSCAN 算法不需要提前确定簇类的数量，而且 DBSCAN 可以发现任意形状的聚类。但是 DBSCAN 也存在不足，如图 7-22 所示，虚线表示正确聚类，但是由于其中部分数据密度过高，容易导致异常聚类。

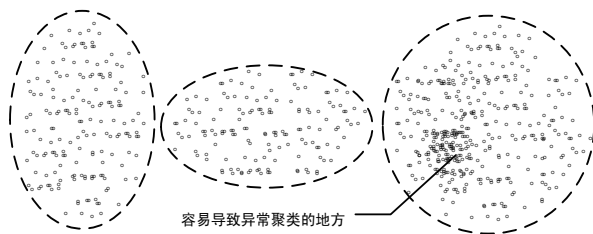


图 7-22 DBSCAN 聚类缺陷示例

缺陷主要集中在两方面：一方面是不能很好地反映高维数据；另一方面是在聚类密度不断变化的数据集中，不能很好地反映整体聚类情况。

对于搜索引擎中的知识聚类，针对 DBSCAN 算法，将知识实体的名称作为每个点，点与点之间的距离被认为是语义相似度，其次需要确定半径范围 (eps) 和半径范围内的数量 (MinPts)。在知识图谱中对实体聚类，有助于实体模块化，在快速检索及查找过程中，可以在聚类簇中搜索，而不必对整个知识图谱

进行查找，不采用分类方法的原因是，一个实体可能属于多个分类，而且分类界限不明晰，通过聚类以其自然特征的方式聚合在一起显得更加合理。

7.9 智能搜索实现

知识图谱以结构化的方式进行存储，而用户的搜索是使用自然语言，更不可能让用户使用编程的方式去使用。因此，将用户的自然语言转换为内部系统可识别的检索指令也是知识图谱检索的核心任务之一，简单来说即是将用户的自然语言转换为系统可识别的 N 条查询语句。主要通过模式匹配、知识拆解、合并求解三个层次渐进分析。

(1) 模式匹配。并不是所有用户的搜索都会对知识图谱进行一次查询，因此通过匹配用户的搜索词，确定是否存在进行知识图谱查询的需要。

(2) 知识拆解。匹配到搜索词时，需要将其搜索信息拆解。拆解为各个子知识搜索，将子知识搜索进行知识图谱查询。

(3) 合并求解。子知识通过知识图谱返回的结果，需要将其合并为用户查询的解，并将结果返回给用户。

7.9.1 模式匹配

模式匹配需要较好的实用性，针对知识图谱和人工智能领域，人工智能标记语言（Artificial Intelligence Markup Language, AIML）拥有较好的模式匹配思想。通过 AIML 可以将用户搜索词转化为系统内部查询指令。AIML 是著名的人工智能机器人 ALICE（Artificial Linguistic Internet Computer Entity）的知识库存放形式。一个完整的 AIML 文件类似于如下代码表示：



```
<?xml version="1.0" encoding="gb2312"?>
<aiml>
  <category>
    <pattern>你 叫 什么 名字 *</pattern>
    <random>
      <li>我叫Alice</li>
      <li>我的名字是Alice</li>
      <li>Alice</li>
      <li>我是Alice</li>
      <li>我是大名鼎鼎的Alice机器人</li>
    </random>
  </category>
  <category>
    <pattern>你 的 名字 *</pattern>
    <template>
      <that>你 叫 什么 名字 呢</that>
    </template>
  </category>
</aiml>
```

AIML 实质上是一个 XML 配置文件,每个智能问答模板都是通过“category”进行描述的。例如,问机器人“你叫什么名字呀?”,命中相应的“pattern”,它会从“random”里面随机选择一个告诉你。为了避免重复性标记,当问机器人“你的名字是什么?”问题时,会自动将问题转换为“你叫什么名字呢”从而和第一个问题一样。

7.9.2 知识拆解

AIML 还有很多特征标记,利用 AIML 模式匹配思想,是为了将用户的搜索转换为知识图谱可识别的查询语言,达到知识拆解的目的。

(1) 需要定义知识图谱查询语言。系统以 JSON 的数据格式定义知识图谱查询语言。笔者定义查询格式如下:

```
{
  "return_type": "",          //两种格式,recursive表示递归,result 表示结果
  "knowledge_query": [ ]     // 向知识图谱发起的请求信息,包含数据可以是多条
```

```

"result_express": "",          // 结果返回的表达式
"result_script": " ",         // 执行脚本
"debug": ""                  // 调试信息
}

```

(2) 与人工智能标记语言结合。将 AIML 中的 `pattern` 变更为需要匹配的知识信息，将 AIML 的回答模板变更为知识图谱查询语言，如下代码所示：

```

<category>
  <pattern>
    *|nr 比 *|nr 高 吗 *
  </pattern>
  <template>
    {
      "return_type": "recursive",
      "knowledge_query": [
        {
          "command": "knowledge",
          "entity": "<star index='1' />",
          "parameter": "身高"
        },
        {
          "command": "knowledge",
          "entity": "<star index='2' />"
          "parameter": "身高"
        }
      ],
      "result_express": "#s1#-#s2#",
      "result_script": "if(\"#result_express#\".contains(\"unknown\"))
{return \"未找到可以比较的数据。\";};if(!\"#result_express#\".
contains(\"=-\")){return \"是的，高#result_express#\";}return \"不是的，
相差#result_express#\";\",
      "debug": "拆分信息'<star index='1' />身高'、'<star index='2' />身高'"
    }
  </template>
</category>

```

(3) 实现知识拆解。倘若搜索“刘翔比刘德华高吗？”，首先会通过分词并作词性标注得到“刘翔|nr 比|p 刘德华|nr 高|a 吗|y？|w”，在上述“`pattern`”中，允许两个任意符匹配，但是要求它们匹配到的词性必须是“nr”（人名）。



“<star index='1'/>”表示指代第一个“*”所指内容。通过匹配会替换“knowledge_query”信息如下，有效地将搜索“刘翔比刘德华高吗？”转变为两个问题“刘翔身高”和“刘德华身高”，而两个子知识问题，可以通过知识图谱直接求解。

```
{
  "command": "knowledge",
  "entity": "刘翔",
  "parameter": "身高"
},
{
  "command": "knowledge",
  "entity": "刘德华",
  "parameter": "身高"
}
```

上述示例中，在“pattern”里限定了匹配“*”必须是人名，否则去查询实体的“身高”就失去意义。例如，用户搜索“武夷山比五台山高吗？”，通过分词与词性标注“武夷山|ns比|p五台山|ns高|a吗|y？|w”，“ns”表示地理名，地理名称则应当使用“海拔”，只需新增相关“category”即可，如果没有命中相应的“category”则直接搜索网页即可，跳过知识图谱的搜索。

7.9.3 合并求解

在知识拆解之后，会通过子知识在知识图谱中搜索答案。接前面示例，通过知识图谱并发查询，获得刘翔身高189 cm，刘德华身高174 cm。合并求解过程，需要通过表达式生成、表达式计算、结果返回三部分完成。

(1) 表达式生成。在定义的查询语言中，存在字段“result_express”，即表达式生成模板。子知识中查询到的结果会填充到“result_express”: “#s1#-#s2#”中，即：

```
"result_express": "189cm-174cm"。
```

(2) 表达式计算。在定义的查询语言中，存在定义字段“return_type”，表示返回数据类型。如下代码所示，表示这是一个递归运算。


```
"return_type": "recursive"
```

此时需要将“result_express”进行递归，也就是将“189 cm-174 cm”进行再次查询转换，它们会命中另外一个 category。

```
<category>
<pattern>
*|m cm <set>加减乘除</set> *|m cm
</pattern>
<template>
{
  "return_type": "result",
  "knowledge_query ": [],
  "result_express": "<star index='1' /><star index='2' /><star index='3' />",
  "result_script": " return \"#result_express# (单位: 厘米) \";",
  "debug": "推理过程:<star index='1' /><star index='2' /><star index='3' />"
}
</template>
</category>
```

此 category 返回的结果是 result，因此不需要再次递归，“<set> 加减乘除 </set>”表示“+ - * /”任一符号，因此 "result_express" 为“189-174”，通过计算表达式，结果为 15，即 #result_express# 为 15。

(3) 结果返回。在定义的查询语言中，字段“result_script”表示返回结果定义脚本。因此，通过代码执行 "result_script" 直接返回“15（单位：cm）”，此结果将成为上一个“category”的 "result_express"，因此上一个 category 的“result_script”为：

```
if(!"15（单位：cm）\".contains(\"unknown\"))
{
  return \"未找到可以比较的数据。\";
};
if(!"15（单位：cm）\".contains(\"=-\"))
{
  return \"是的，高15（单位：cm）\";
}
return \"不是的，相差15（单位：cm）\";
```



通过执行“result_script”，最终用户搜索“刘翔比刘德华高吗？”，系统将返回“是的，高 15（单位：cm）”。

7.10 智能搜索扩展

通过将人工智能标记语言（AIML）作为模式匹配，以及知识递归推理得到最终准确答案，用户的任何一个搜索均可通过自定义 AIML 的模式匹配，进行拆解信息，分块获取信息，然后将分块知识再合并，最终求解给出答案。值得说明的是，采用 AIML 除作为语法解析的中转、知识递归逻辑推理之外，还可以对智能搜索进行扩展。

7.10.1 常识性智能搜索

在实际工程中，许多常识性、固定化的信息是无须通过互联网学习即可拥有的。例如搜索“每周有多少天？”，此类搜索只需通过配置人工智能标记语言即可，如下代码所示，根据“random”中的数据，随机返回即可。

```
<category>
  <pattern>每周有多少天 *</pattern>
  <random>
    <li>七天</li>
    <li>每周有七天</li>
    <li>一周共七天</li>
    <li>七天为一周</li>
    <li>从周一到周日，共七天</li>
  </random>
</category>
```

类似还包括“闰年一年多少天”“中国有多少个省份”“农历节日”等，通过人工智能标记语言直接返回答案，无须对知识图谱请求，在性能和用户体验上都可以达到很好的效果。

7.10.2 实时信息智能搜索

人工智能标记语言很好地通过知识图谱返回相关知识，但是存在另外一种情况，知识图谱很难包含实时信息，如每天的天气变化情况、时刻变化的航班信息等，即使通过知识图谱去获取，那么时效性也不能得到保证。因此可以通过查询语言定义的“knowledge_query”灵活配置 command 以解决此问题。例如搜索“北京今天天气”，配置信息如下，所有城市的“今天天气”的搜索都会经过此配置。

```
<category>
  <pattern>
    *|ns 今天 天气 *
  </pattern>
  <template>
    {
      "return_type": "result",
      "knowledge_query": [
        {
          "command": "weather",
          "entity": " <star index='1' />",
          "parameter": "今天"
        }
      ]
      "result_express": "",
      "result_script": "";",
      "debug": "城市 '<star index='1' />' ， 查询日期: 今天"
    }
  </template>
```

</category> 在天气请求中，直接将“command”的值变更为“weather”，传递相应参数即可，当然也可以是航班号、彩票等其他变化性较强的搜索信息，关键在于后台根据收到的“command”能够有效获取数据。天气、航班、彩票等大多数都属于开放性数据，可通过开放接口直接获得。

7.10.3 可交互式智能搜索

最初研发的目的是人工智能标记语言（AIML），用在著名聊天机器人



ALICE 上面, 因此, 将 AIML 原始的功能保留在现有搜索引擎系统中也是可以的, 只需做出相应的配置, 对于搜索引擎来说, 即是交互性搜索。

```
<category>
<pattern>
<set>美食菜谱</set>的做法
</pattern>
<template>
{
    "return_type": "result",
    "knowledge_query": [
        {
            "command": "cookie",
            "entity": "<star index='1' />",
            "parameter": "做法"
        }
    ]
    "result_express": "",
    "result_script": "";",
    "debug": "查询 '<star index='1' />' 的做法 "
}
</template>
</category>
<category>
<pattern>
<set>美食菜谱</set>
</pattern>
<template>
您是想知道<star index='1' />怎么做吗
</template>
</category>
<category>
<pattern>
是的
</pattern>
<that>您是想知道 * 怎么做吗</that>
<template>
<srai>
<thatstar />的做法
</srai>
</template>
</category>
```

上述示例的 AIML 配置的信息是：当用户搜索“宫保鸡丁”，则搜索引擎除返回“宫保鸡丁”相关网页外，还会提示用户“您是想知道宫保鸡丁怎么做吗？”，如果用户回答“是的”，则通过知识图谱返回“宫保鸡丁”的详细做法。

可以发现根据人工智能标记语言重新定义的智能查询语言的强大不仅仅在于知识图谱的信息拆解，还在于灵活的交互能力，人工智能标记语言将会成为未来人工智能发展的基础。

7.11 本章小结

知识图谱是现代搜索引擎智能化程度的标识，它不仅仅是一次技术的更新，更是下一代搜索引擎的基础。据不完全统计，Google 目前的知识图谱至少包含了 5 亿实体，构建强大而又完整的知识图谱，可以为网民大众提供直接答案以减少用户在搜索结果页中的查找时间，协助搜索引擎提升用户搜索体验。知识图谱通过知识发现、知识推理、知识计算、知识聚类后将单一的知识串联为信息价值，不断完善知识图谱体系结构，实现知识图谱价值最大化，使得知识图谱最大限度地为不同人群服务。

第 8 章 索引构建机制

搜索引擎的索引类似于人的反应中枢，数据存储结构直接关系到搜索引擎的查询性能。在大数据时代，搜索引擎的数据存储量呈现指数增长，而索引往往会比原始数据大很多，这给当代的索引构建带来了巨大挑战，原因在于一方面要保证索引存储大小能够有效控制；另一方面则能够通过索引快速找到相关数据。能够通过索引快速找到相关数据取决于如下两方面：

（1）索引结构设计。合理的索引结构，可以保证在相同环境下，在尽可能短的时间复杂度范围内获得相应数据。

（2）硬件系统资源。硬件资源也是非常重要的，例如采用 SSD 的硬盘系统，相同条件下处理性能一定优于普通硬盘。

硬件系统可以通过购买新硬件解决，但索引结构设计则是对开发人员非常重要的一块。一个完整的搜索引擎至少包含两类索引：倒排索引和字典树索引，分别用于检索和智能提示（又称为自动联想）。

8.1 倒排索引

平常使用的大多是基于数据库的索引。而搜索引擎使用倒排索引（Inverted Index），它是全文搜索中最常用的数据存储结构，被用来存储文档集合中某个

单词对应的文档集合及单词在文档中存在的位置，因此又称为反向索引。

8.1.1 倒排索引概述

与反向索引相对应的是正向索引。正向索引是存储每个文档的单词列表，并记录其位置，以表 8-1 中句子为例。

表 8-1 示例句子

文档编号	句子内容
文档_1	你好，搜索引擎
文档_2	搜索引擎技术窥视
文档_3	你好，搜索技术
.....

对表 8-1 中的句子建立正向索引，如表 8-2 所示。

表 8-2 正向索引存储示例

文档编号	关键词及位置
文档_1	你好 (0)，搜索引擎 (2)
文档_2	搜索引擎 (0)，技术 (4)，窥视 (6)
文档_3	你好 (0)，搜索 (2)，技术 (4)
.....

对表 8-1 中的句子建立倒排索引，如表 8-3 所示。

表 8-3 倒排索引存储示例

关键词	文档编号
你好	文档_1 (0)，文档_3 (0)
搜索引擎	文档_1 (2)，文档_2 (0)
技术	文档_2 (4)，文档_3 (4)
窥视	文档_2 (6)

表 8-3 是根据文档生成的倒排索引，当用户搜索“你好搜索引擎”时，会将关键词对应的文档集合取出，并且会根据相关性对文档进行排序处理。



在某些搜索引擎系统中，可能会有不存储单词在文档中位置的情况，但是目前大多数搜索引擎都会存储，主要基于两个原因：一是大数据时代背景下，全文检索过程中，如果没有记录位置，在获取文章动态摘要时，会通过遍历文档的方式查找关键词出现的位置，存在性能损耗，而这对于数据量级较小的引擎则影响较小；二是在搜索结果展示过程中，为获得较好的动态摘要，需要对文档内搜索词分布情况进行统计，一般情况下会在较集中关键词的区域提取摘要，如果提前确定关键词位置，对搜索体验会有很好的提升。

8.1.2 索引结构

倒排索引是搜索引擎算法中最重要的数据结构之一，它可以在极短的时间内定位文档的具体位置。构建倒排索引最重要的是设计合理的数据存储结构，如果仅仅是按照传统的方式构建，至少会有如下缺陷：

（1）支持索引容量较小，占用存储空间较大。工程实践表明，索引数据远远大于搜索引擎抓取网页的数据大小。网页数据存储于物理硬盘，倘若将索引数据放入内存，将会导致内存消耗较大，尤其是在大数据时代，优秀的数据结构可以帮助企业减少成本。小型搜索系统或垂直搜索常常将索引数据直接放入内存，一方面便于管理，另一方面便于查询，但对于大数据时代来说，则是一种不可取的方法，也不利于业务长期发展。

（2）排序可靠性较差。词语在文档中出现，仅仅说明词语与文档存在关系，若是依靠出现关系，则在搜索排序中，会产生很多杂乱无章的文档数据，因此这样的可靠性值得质疑。搜索结果的相关性排序，不仅仅依赖于词语的出现，还取决于词语的重要性，并不是所有词语对网页同等重要。

基于上述两点，需要对倒排索引数据结构做一次改进，如图 8-1 和图 8-2 所示。

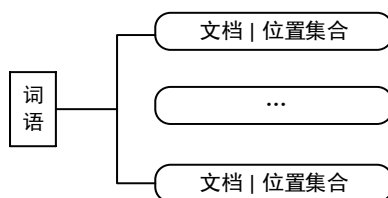


图 8-1 传统倒排索引数据结构

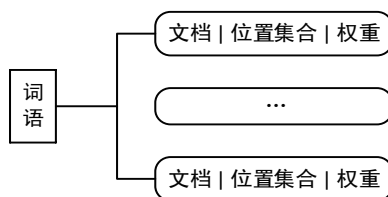


图 8-2 改进倒排索引数据结构

通过存储结构对比，图 8-2 在存储数据中比图 8-1 多权重一项，权重是指词语对于当前文档的重要性，单词的权重通过 TextRank 计算得出（算法见 3.6 节）。

增加数据存储结构的方法有效地避免了相关性排序可靠性较差的问题，但是支持索引容量较小问题反而更加严重。从工程应用角度分析，假设某垂直领域搜录网页数千万，那么建立的倒排索引中，每一个词语都有可能包含数万甚至数十万网页，将这样庞大的倒排索引表写入内存，是非常不合理的，笔者曾经用这样的方式搜录一千万的网页量时，用多台机器内存协同加载倒排索引，内存都基本消耗殆尽。因此需要在倒排索引之上，再建立一层二级索引，如图 8-3 所示。

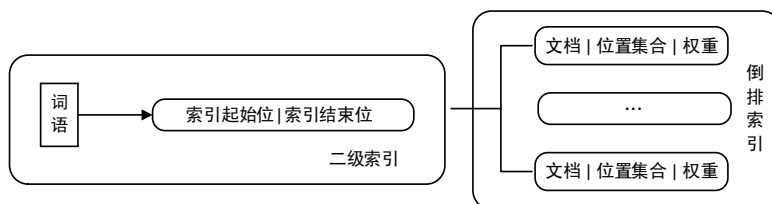


图 8-3 基于哈希表的二级索引

建立二级索引的方法有两种：通过哈希表及 B^+ 树，图 8-3 所示为通过哈希

表建立二级索引。通过将哈希表加载到内存中，以词语作为哈希表的键，值为单词在倒排索引中的位置信息，用户的搜索请求先通过二级索引，找到对应词语倒排索引存储的起始和结束为止，通过起始和结束位置，读取位于硬盘上对应的倒排索引即可。 B^+ 树索引，则类似于数据库中的文件索引，针对海量数据二级索引的构建采用 B^+ 树的方式更加合理，可参考 8.3 节详细介绍。

8.1.3 构建过程

在第 6 章中，分布式爬虫已经从互联网中提取到各类数据，倒排索引的构建过程即是各类数据进行索引优化的过程，如图 8-4 所示。

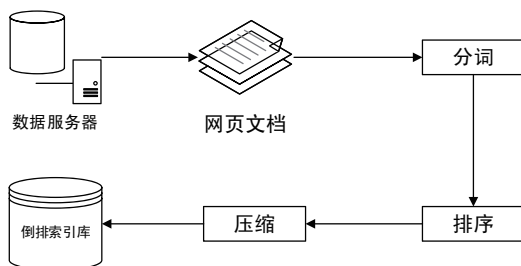


图 8-4 倒排索引构建过程

（1）提取文档。将文档从大数据存储引擎（如 HBase）中提取出来，文档信息包括网页标题、网页正文、网页链接、网页作者等相关信息，均为爬虫整理收集分析的结构化数据。

（2）文档分词。对文档进行分词处理，分词过程包括停用词移除、中文分词、英文分词等，并且建立正向索引。

（3）词权重计算。对正向索引中的词进行权重计算。

（4）索引排序。根据正向索引生成倒排索引，并对倒排索引中的内容进行排序，包括对词排序和文档列表排序。

(5) 压缩入库。构建二级索引并对索引数据进行压缩, 将压缩之后的倒排索引存入索引库。

上述过程均是顺序执行, 每个阶段都会有大量并发节点在运行, 整个过程中存在一个缺陷, 它需要足够的内存来存储倒排索引, 因此, 在构建过程中常常需要利用归并法, 当内存中倒排索引量达到一定限制时, 将其写入文件, 生成临时倒排索引文件。然后对不断生成的多个临时倒排文件, 利用多路归并方式, 得到最终的倒排索引文件。

8.1.4 排序规则

为增强倒排索引检索效率, 除对结构优化之外, 还需要对其各个数据项进行排序。二级索引中, 记录了词语的索引起始位置和结束位置, 然而并不是所有词语在构建索引过程中, 起始位置和结束位置都是该词语的倒排信息, 中间还夹杂着其他词语, 词语在倒排上的相互交叉性比较多。其次, 即使所有词语的倒排索引都在一个区域, 这个区域内词语的权重不一定是有序的。因此, 索引排序包含词语排序和文档排序。

(1) 词语排序。是指将索引中相同的词语集中放置到一个区域, 将零散的集合变为有序的集合, 便于数据检索而不必遍历整个倒排索引, 排序后效果如图 8-5 所示。

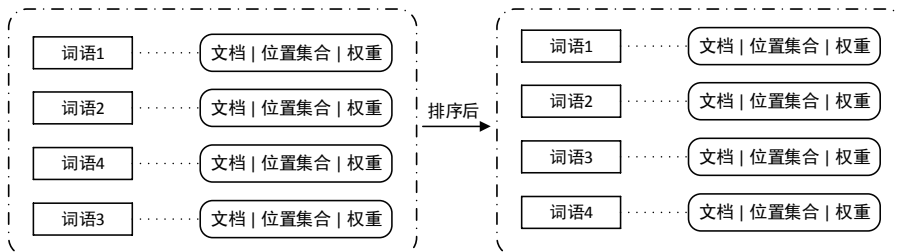


图 8-5 倒排索引词语排序

(2) 文档排序。文档排序是针对具体词语映射的文档集合，将每一组词语的倒排索引内部对应文档集合按照权重从高到低进行排序，如图 8-6 所示，同一词语中，词语所在文档权重较高的文档出现在前面。

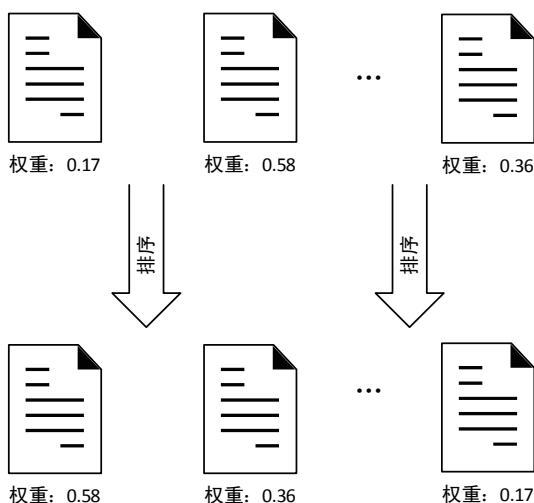


图 8-6 倒排索引中权值排序

针对词语的权值进行排序便于在检索时直接提取出词语有序的集合，而不必每次检索时，都需要对词语进行一次排序。

8.1.5 索引压缩

结合工程应用角度，倒排索引非常庞大且远远超过网页本身大小，在不影响快速检索的情况下，需要对倒排索引进行压缩，以减少内存和硬盘开销。可以通过文档编号差值化、词语哈希化、文档拆减等方式进行压缩。

(1) 文档编号差值化。文档编号差值是倒排列表中相邻的两个倒排索引项文档编号的差值，在权值有序的文档中，文档编号差值可能存在负值。图 8-7 所示的例子中，原始的 5 个文档编号分别是 200、208、196、270 和 150，通过编号差值计算，在实际存储时就转化成了：200、8、-4、70 和 -50。

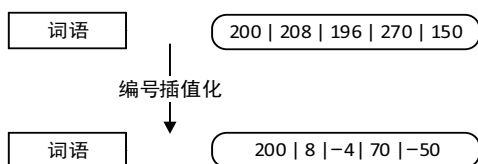


图 8-7 倒排索引中有效的文档编号差值化处理

文档编号差值化则表示基于第一个元素，将后面的文档编码按照与第一个文档的差值进行存储。当然并不是所有索引经过差值化都可以达到压缩的目的，图 8-8 所示为无效的文档编号差值化处理。然而对于庞大的索引数据集，通过文档编号差值化可以有效减少存储大小。

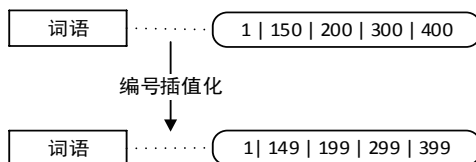


图 8-8 倒排索引中无效的文档编号差值化处理

因此，对文档编号进行差值计算，是为了更好地对数据进行压缩，原始文档编号一般都是大数值，通过差值计算，就有效地将大数值转换成了小数值，而这有助于增加数据的压缩率。实际应用中，也有对词语在文档中位置的差值化，方法同上。

(2) 词语哈希化或哈夫曼编码。无论是在内存还是硬盘存储过程中，词语都会存在，然而词语量依然非常庞大，普通的应用程序开发可能如图 8-9 所示。

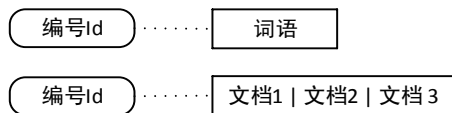


图 8-9 倒排索引中词语进行编号设计

将每一个词语对应一个编号，利用生成的编号去和文档建立对应关系，如果平均每个词语对应 10 万个文档集合，那么该词语将会减少 10 万次存储，转

而用数值代替。数值存储会远远小于字符串存储，但是实际过程中，不会存在一个词语和编号对应的关系表，一方面是耗费空间，另一方面是在中文分词中存在新词识别，如果新词不存在编号，则需要给予生成编号。因此，从工程学角度来看，可以利用词语的哈希化解决此类问题，类似于生成词语编号对应关系，只是哈希化是不可逆的。

除进行词语哈希化之外，还可以通过哈夫曼编码方式对词语进行数值化。根据对词语进行哈夫曼编码处理，可减少网络传输流量。哈夫曼编码是一种无损数据压缩的权编码算法。哈夫曼编码的思想是：通过变长编码的方式对原始数据进行编码，其中的变长编码表是通过权值评估的方式获得的，出现权值较高的词语具有较短的编码，反之权值较低的词语具有较长的编码，使得整个数据的平均传输长度变短，从而达到无损压缩数据的目的。

针对关键词，可以通过权重的关系将频繁出现的词语给予较短编码，权重则以词频体现。例如，通过对历史数据进行离线分析，得到所有词与词频的关系，如表 8-4 所示。

表 8-4 利用哈夫曼树进行词语编码的词频统计信息

编 号	词 语	词 频
01	北京	3000
02	俄罗斯	4000
03	普吉岛	2000
04	空旷	2000
05	沃尔沃	2000
06	儒家	1000
07	感激	1000

将表 8-4 中的词语按照词频由低到高排序，如图 8-10 所示。

儒家 1000	感激 1000	普吉岛 2000	沃尔沃 2000	空旷 2000	北京 3000	俄罗斯 4000
------------	------------	-------------	-------------	------------	------------	-------------

图 8-10 将需要进行哈夫曼编码的词语按照词频排序

在将图 8-10 中的词语进行哈夫曼编码之前，需要建立哈夫曼树。哈夫曼树又称为最优二叉树，是一种带权路径长度最短的二叉树。带权路径长度是指所有叶节点的权值乘以叶节点到根节点的长度，将所有的叶节点的带权路径长度相加，得到的值是最小的。这也是哈夫曼树编码根据权重编码后能够达到效果的理论根据。

① 选择其中权值最低的两个词语，组建一个二叉树，如图 8-11 所示。

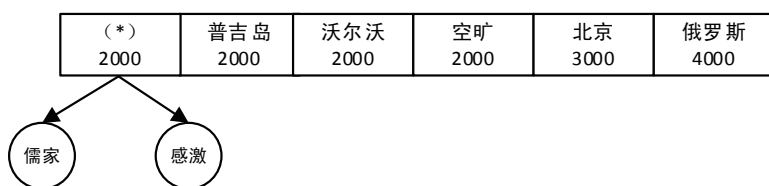


图 8-11 哈夫曼树构建过程中将最小词频词语组成二叉树

② 将图 8-11 中新组成的二叉树当作整个列表中的一个元素，元素权重等于两个词的权重之和。然后再次重复第一个步骤，选取两个权值最低的词语组建新的二叉树，如图 8-12 所示。

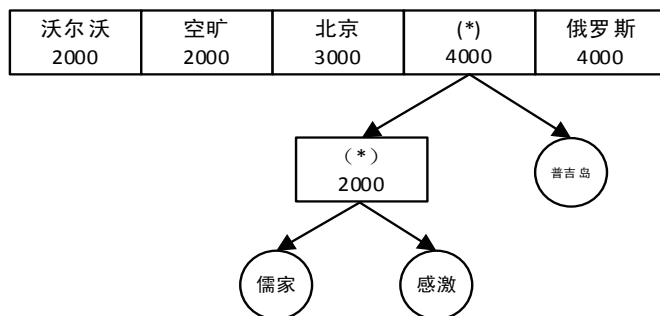


图 8-12 将最小词频的词语再次加入到哈夫曼树中

③ 依次重复迭代上述过程，直到所有的词汇都成为二叉树中的元素，如图 8-13 所示。

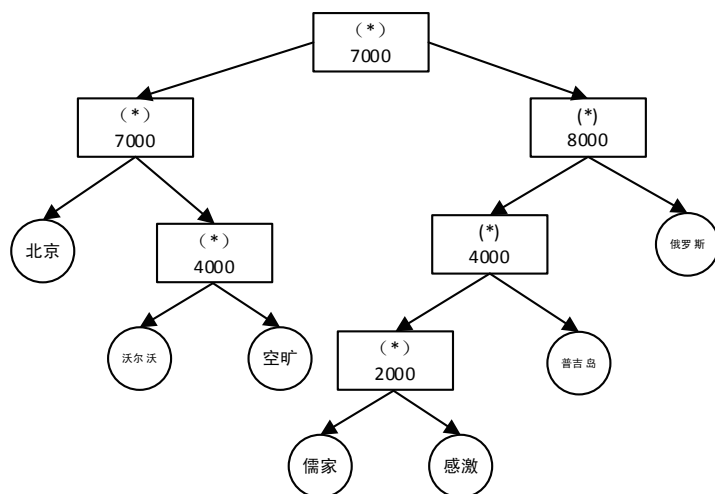


图 8-13 根据词频构建的哈夫曼树

图 8-13 已经成功构建了哈夫曼树，只需将上述哈夫曼树中所有右子树的边设定为 1，所有左子树的边设置为 0，如图 8-14 所示。

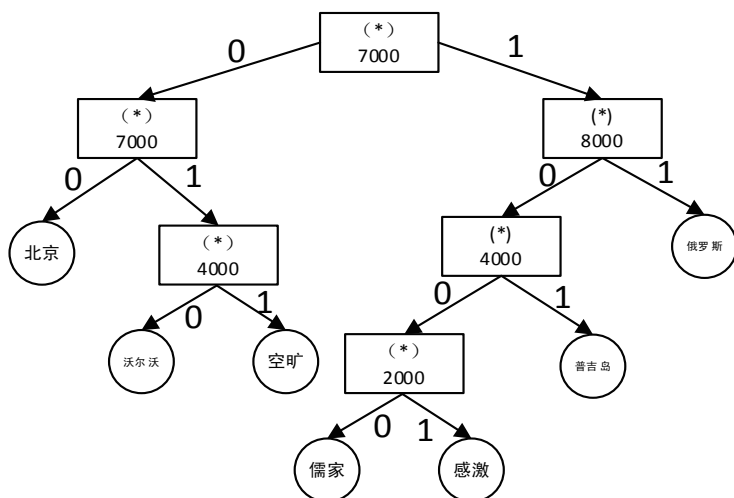


图 8-14 根据哈夫曼树标记哈夫曼编码值

根据图 8-14，需要获得每个词语的哈夫曼编码，只需获取根节点到各个词语之间的路径即可，最终获得各词语的哈夫曼编码如表 8-5 所示。

表 8-5 根据词频最终获得各词语的哈夫曼编码

编 号	词 语	词 频	哈夫曼编码
01	北京	3000	00
02	俄罗斯	4000	11
03	普吉岛	2000	101
04	空旷	2000	011
05	沃尔沃	2000	010
06	儒家	1000	1000
07	感激	1000	1001

通过表 8-5 可以得出结论：高词频词语通过哈夫曼编码之后的长度较短，而词频较低的词语经过哈夫曼编码之后的长度较长，从而达到无损压缩的目的。

词语的哈希化和哈夫曼编码方式都可以达到词语压缩的目的，相对来说，哈夫曼编码达到的效果更好，压缩率更高，但是由于哈夫曼编码方式需要对词语提前进行离线计算。在工程应用中，会根据实际情况，选择词语的哈希化或者哈夫曼编码进行词语数值化。

(3) 文档拆减。文档拆减顾名思义是减少文档数量，有两种方式。一种是通过权重极低值拆减：在现代搜索引擎中，用户访问的页面量是极少的，即使返回 100 条搜索结果，用户也许就关心靠前的 20 条结果，对于词语对应的权值极低文档，可以直接移除掉，因为无论采用何种搜索算法，都无法使得其排序进入 100 条搜索结果范围内。具体移除的数值取决于返回的结果数量。另一种是通过过滤词拆减：在索引过程中，并不是所有的词语都需要建立倒排索引，对于某些词语，例如“的”“啦”“a”“as”“of”“the”等，俗称停用词，这类词语并不需要建立倒排索引。停用词一般属于文档中基本上都会用到的词语，它对应的文档数量集合庞大，然而对实际搜索排序却没有实际意义。这类词语在中文分词的时候，尽量过滤掉，以避免其影响搜索结果和搜索性能。



8.1.6 更新策略

倒排索引是一种静态数据集合，而互联网上庞大的信息却是动态的。搜索引擎在理论上存在两种索引更新策略，分别是完全更新策略和合并更新策略。

(1) 完全更新策略。几乎利用索引版本将过去的索引完全替换掉，无论索引中发生变化的内容多或者少，新索引全部替代旧索引。此类方法简单彻底，对于中小型索引是可取的，但是对于十亿级的网页索引进行完全更新，显得不够现实，代价过大。

(2) 合并更新策略。是将新增网页的索引与已经存在网页的索引进行合并，可行性比较高，性能代价较小，但存在的问题是，如果不是新增网页，而是对已经存在的网页索引进行更新，则将会付出较大代价。因为某一篇文章的更新，导致以前文档中的词语需要在索引中移除，然后插入新的索引值。

对于索引的更新难点在于索引数据删除，更新实质上是将上一次该文档的数据删除，然后通过插入新索引数据完成。从用户体验的角度来讲，一方面对搜索引擎索引到的数据应该尽可能快地搜索到，而不是一天甚至一周之后才被搜索到；另一个方面，已经被删除的页面，链接已经无效，则应该禁止该链接被检索出，即使与用户的搜索具备一定相关性，由于链接已经失效，用户点击也无法正常打开页面，严重影响了用户体验。

8.2 分布式存储

索引压缩拆减只是相对减少了数据，在当前大数据时代，索引数据会超出想象的庞大。因此，采用分布式存储方式有利于数据保护和处理。

8.2.1 存储划分方式

分布式存储可以按照基于文档进行分布式存储和基于词语进行分布式划分。

(1) 基于文档进行分布式存储。每台分布式服务器分别存储不同文档编号区间的索引，则意味着每台服务器维护的倒排索引文档编号互不相同，如图 8-15 所示。

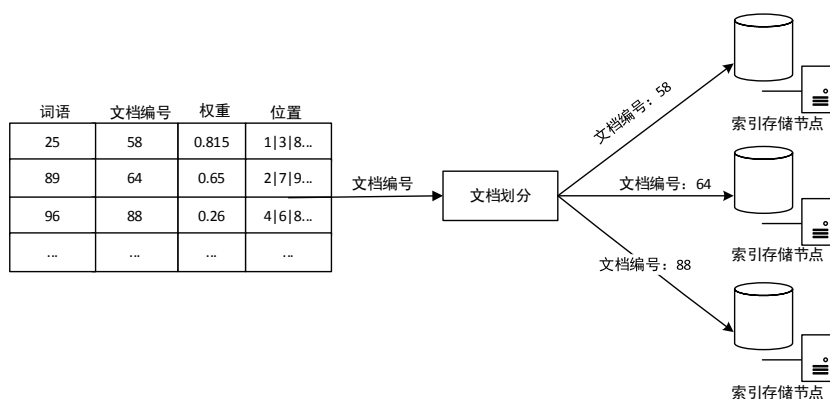


图 8-15 基于文档的索引分布式存储

在检索过程中，会以广播的形式将搜索词发送到每台索引存储节点获得对应的倒排文档集合，然后将所有倒排文档合并。

(2) 基于词语进行分布式划分。索引过程中按照索引词的不同，分别存储到各个索引存储节点中。每个索引节点存储的词互不相同，但是每个索引服务器中可能包含相同的文档，如图 8-16 所示。

若采用基于词语的划分存储，在搜索过程中，则不需要通过广播向所有的服务器发送搜索请求，对于减少网络请求次数和磁盘 I/O 均有一定好处。但是工程应用中，采用的是基于文档的划分存储，主要基于可靠性和效率来考虑。

基于词语进行分布式划分的缺点如下：

①基于词语划分无法保证单点故障。若在索引服务器集群中，某一服务器

发生故障，则很可能导致该服务器上对应的词语都无法实现正常搜索，甚至无结果返回。而基于文档的划分至多是这部分文档无法被搜索出，其他文档依然可以正常被检索。

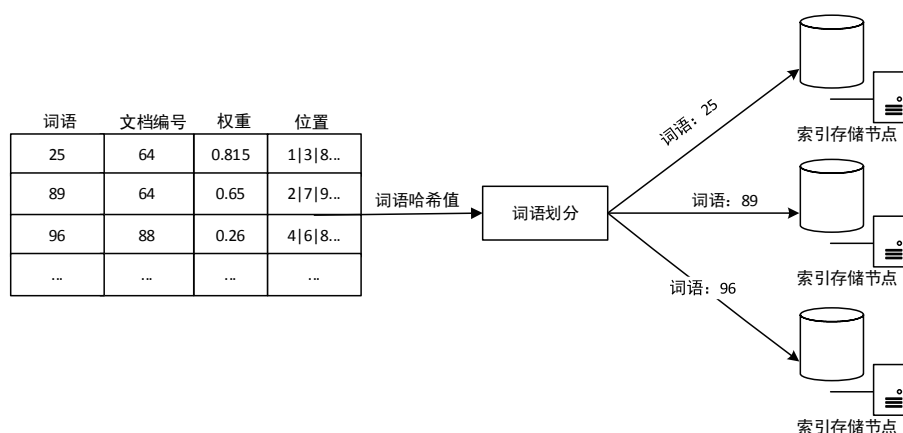


图 8-16 基于词语的索引分布式存储

② 基于词语划分方式并行处理效率较低。搜索结果的索引如果仅来自某台索引服务器，则整个检索过程中，都会等待该索引服务器返回数据。而采用文档的划分存储，索引数据的获取由各个索引服务器并发共同完成。

8.2.2 存储平衡策略

分布式处理的方法存在很多种，一般采用 N 台服务器作为分布式存储集群，当进行数据存储时，只需将需要存储的文档进行哈希运算得到哈希值 V 。对于 N 台服务器，则对应数据即存放在 V 对 N 求余数的服务器上，理论上这种方式是可行的，而且结构简单，易于实现。但是工程应用中，出于种种原因，却无法胜任，原因如下：

(1) 扩展性差。随着业务规模和用户群的不断壮大，不得不希望分布式系统能够存储更多缓存信息，但是一旦扩大分布式索引规模，原始的所有索引就

会受到影响，每次对集群修改都会存在极大影响，对于任何在线服务均是不允许的。

(2) 容灾性较差。服务器难免会宕机、机房也存在停电的风险，倘若索引集群中的一台服务器发生异常，那么该服务器上的索引数据无法使用，搜索引擎需要尽可能避免这样的事情发生。

(3) 健壮性差。服务器更改配置进行索引数据迁移，极具困难和挑战性。

因此分布式索引存储中，需要集中解决上述问题，实质是解决数据平衡、持久服务问题。

(1) 解决数据平衡问题。将所有索引数据尽可能平均地分配到缓存集群中，避免某台服务器缓存任务过重，也避免过轻。

(2) 解决数据哈希位置不变问题。如果已经有旧的索引数据存在，在新的索引合并到系统中后，哈希算法的结果应该保证现有索引数据依然能够正常使用，而服务不被终止。

在分布式集群中，对集群节点的新增、移除已经是集群管理最基本的管理，因此，常规的哈希算法不能满足分布式索引存储中的要求。

引入一致性哈希算法可以解决上述问题。一致性哈希算法是在 1997 年提出的一种分布式哈希实现算法，其主要思想是将集群中的每一台服务器与一个甚至多个哈希值区间关联，其中，哈希值区间边界通过计算集群中服务器对应的哈希值来确定，如果集群中某一服务器宕机，则它所对应的哈希值区间将会被并入到相邻服务器的区间，而其他服务器保持不变。

一致性哈希算法是将哈希值指定到一个区间，然后将区间首尾相连，组成一个圆环。其次将每个对象映射到圆环上的一个点，对象包括服务器节点，也包括需要存储的索引数据。当存储索引数据时，按照圆环顺时针方向寻找遇到

的第一个索引存储服务器，这台服务器即为当前数据的索引存储服务器。倘若某台服务器发送故障，会将这台服务器上保存的所有对象移至顺时针的下一台服务器中。添加一台服务器，则这台服务器的下一台服务器的部分数据会转移到新的索引存储服务器中，约定哈希值区间 $[0, 2^{32}-1]$ ，表 8-6 所示为五台分布式服务器信息。

表 8-6 五台分布式服务器信息

服务器名（简称 CS）	IP 地址	简 写
Cache Server 1	192.168.1.101	CS 1
Cache Server 2	192.168.1.102	CS 2
Cache Server 3	192.168.1.103	CS 3
Cache Server 4	192.168.1.104	CS 4
Cache Server 5	192.168.1.105	CS 5

对表 8-6 中的服务器构建首尾相连的圆环，如图 8-17 所示。

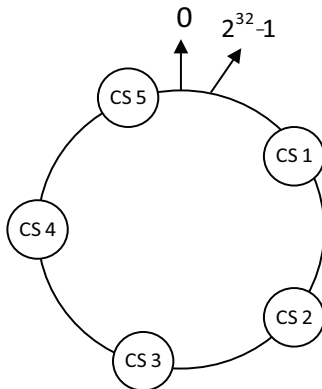


图 8-17 五台服务器组成的环形存储结构

存储方式按照 Key-Value 模式进行存储，对于需要存储的索引数据 “[key1, Value1]” 进行存储，“Key1” 表示被分析的文档标识，“Value1” 表示该文档相关的倒排索引集合。存储过程则只需将 “Key1” 的哈希值对应到圆环上，然后顺时针寻找第一个遇到的节点，该节点即为 “Key1” 索引数据的索引存储服务器。如图 8-18 所示，读取过程和存储过程一致。

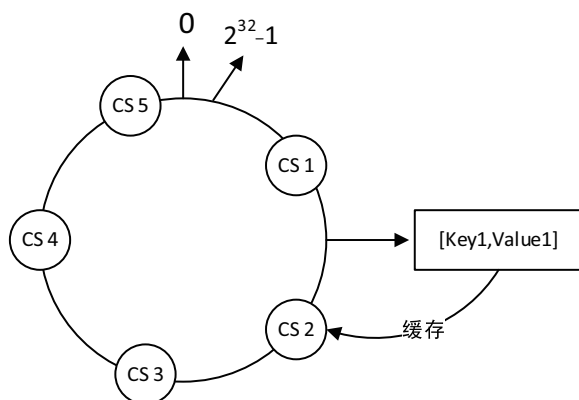


图 8-18 环形存储结构存储示意图

从结构设计角度来看，一致性哈希算法确定的环形存储结构已经具备了较好的容错性和可扩展性。但是也存在不足，如图 8-18 所示，若 CS 2 到 CS 1 的弧线长度（表示数据存放值区间）小于 CS 3 到 CS 4 的长度，这将意味着会有更多的索引数据被放于 CS 4 索引存储服务器上，造成数据倾斜。为解决数据倾斜问题，需要为一致性哈希算法新增虚拟节点的概念，实质是为每个索引存储服务器节点计算多个哈希值，每个哈希值在圆环中的位置均为新的索引存储服务节点，称为虚拟节点，如图 8-19 所示。

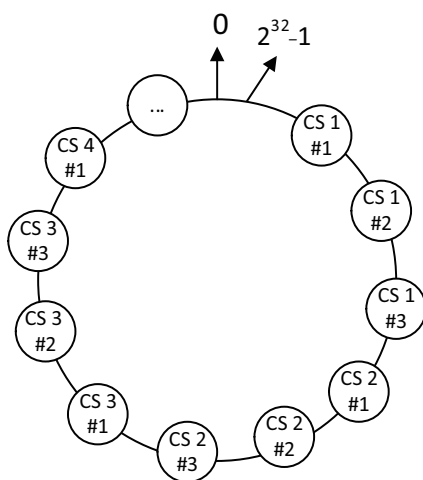


图 8-19 环形存储结构新增虚拟节点

虚拟节点是两个实际节点之间新增的节点，可减少数据倾斜问题。而新增虚拟节点的代价非常小，只需将虚拟节点映射到实际节点，但是带来的效果却不一般。一般真实节点的虚拟节点设置在 32、64 等。

因此，采用改进环形存储结构，用倒排文档的词语作为键，进行分布式服务器选择，选择之后，分别进行存储，如图 8-20 所示。

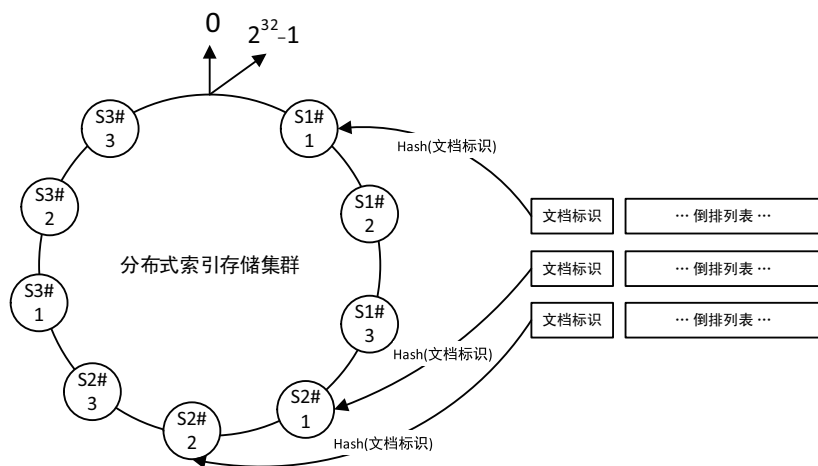


图 8-20 一致性哈希环形分布式索引存储

左边是分布式索引存储服务器，共三台服务器，每台服务器分别构建三个虚拟节点，右侧为倒排索引数据，对关键词进行哈希处理后，将文档集合分别存储在各个节点服务器上。

每台服务器获得的新索引会被以关键词在文档中的权重为值进行归并排序，按照词语的权值由高到低的顺序存储在硬盘中。

在工程中，实际并不会存储所有关键词对应的文档，对于一个关键词对应的文档不会超过 20 万个。原因在于：超过第 20 万后的文档，很难通过排序进入到前 500 的搜索结果（假设返回 50 页，每页 10 个结果）中，也就是说，每个关键词存储的文档，是互联网中与它们最相关的 20 万个网页文档。因此，对

于每台存储服务器来说，实质存储则是排序的两个列表的归并排序问题，并保留前 20 万条记录，由于数据量在内存可接受范围，可以在内存中进行归并排序，如图 8-21 所示。

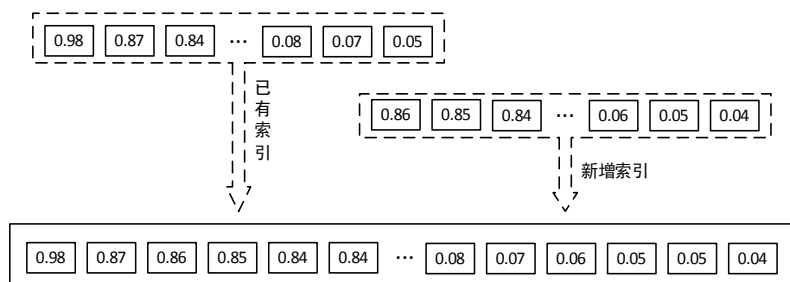


图 8-21 索引的归并排序

归并排序的时间复杂度为 $O(n \ln K)$ ，但并不是新增的文档集合都需要进行插入计算，文档集合中所有文档是按照文档权重进行排序的，倘若从某一条记录开始，已经排满 20 万，则后面记录可以直接忽略。对于文档集合的排序，不需要爬虫服务器或者索引服务器单个运行，采用分布式实时计算，只需将计算结果进行插入即可，相比以往效率提高三倍以上。

二级索引在服务器的内存中，硬盘中是倒排索引生成的倒排文件，二级索引也会在硬盘中生存序列化文件。在搜索过程中，每台服务器将会根据搜索词选取存储服务器，先通过查询服务器上的二级索引，再获得相关文档集合。

8.3 存储索引

倒排索引被分布式存储在各个机器节点中，随着索引文件的增大，索引文本本身的查找性能和顺序扫描也会使得性能下降。倒排索引在各个机器节点中虽然按照一定格式存储，其实质同存储到关系型数据库中类似（不是存储到关系型数据库中），如图 8-22 所示。

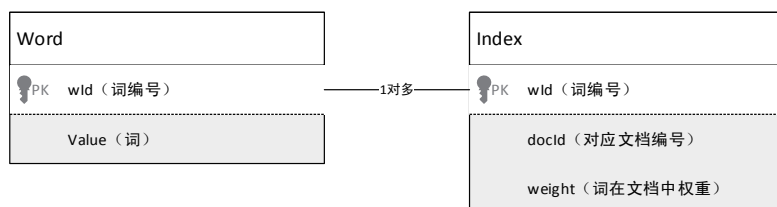


图 8-22 倒排索引在关系型数据库中的逻辑存储结构

根据图 8-22，可以将词语 `wId` 理解为关系型数据库中的主键，读取数据过程只需通过主键快速获得相关数据即可。在关系型数据库中，主键意味着存在与主键一致的索引，而这个索引采用 B^+ 树实现。 B^+ 树是一种树形数据结构，常使用于数据库（如 MySQL、SQL Server 等）和操作系统的文件系统中。采用 B^+ 树索引可以快速获得文件系统中的某些数据。

在介绍 B^+ 树之前，需要先了解 B 树及 B^- 树。

8.3.1 二叉搜索树

B 树是一种改进的二叉搜索树 (Binary Search Tree)，其结构如图 8-23 所示，常常用于存储排序以后的数据，对于树中的数据插入、删除都可以在对数时间内完成。

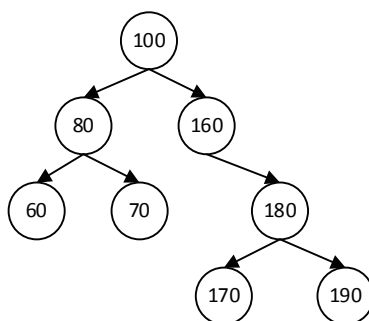


图 8-23 二叉搜索树的结构

它的特征包括：

- (1) 所有节点仅存储一个信息。
- (2) 所有非叶子节点至多拥有两个子结点。
- (3) 每个节点的左节点值一定小于（或等于）当前节点值，右节点值一定大于（或等于）当前节点值。

对于在二叉搜索树上查找一个元素，其性能和二分查找的性能相近，需要 $\log(n)$ 次比较。二叉搜索树的结构不能直接进行数据库索引，原因在于它的查找与树的层数有关系。最差情况下， n 个元素需要进行 n 次查找才能找到目标元素，每查找一层意味着依次读取磁盘，而磁盘 I/O 的性能消耗也远远大于元素查找时间，期望文件索引的目标是访问磁盘的次数越少越好。

8.3.2 B 树

B 树则是对二叉查找树的改进，它的设计思想在于将尽可能相关的元素存放在一起，以便于一次读取更多数据，如图 8-24 所示。

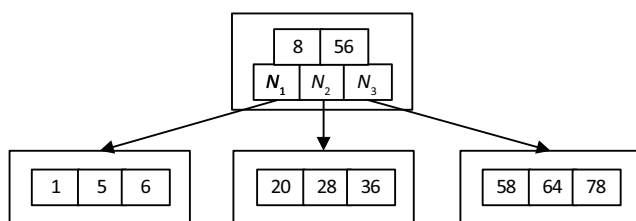


图 8-24 B 树的结构

B 树的特点在于一个节点可以容纳多个值，如图 8-24 中，最多的一个节点存储了 4 个元素。除非当前层的数据已经饱满，否则不会增加新的数据层，因为每增加一层意味着多一次磁盘 I/O 访问。在子节点与父节点的关系中，如果父节点拥有 m 个元素，则其拥有 $m+1$ 个子节点，图 8-24 中，根节点拥有 2 个元素，则其拥有 3 个子节点。

B 树不属于二叉树，是一种多路搜索查找树，如图 8-25 所示。

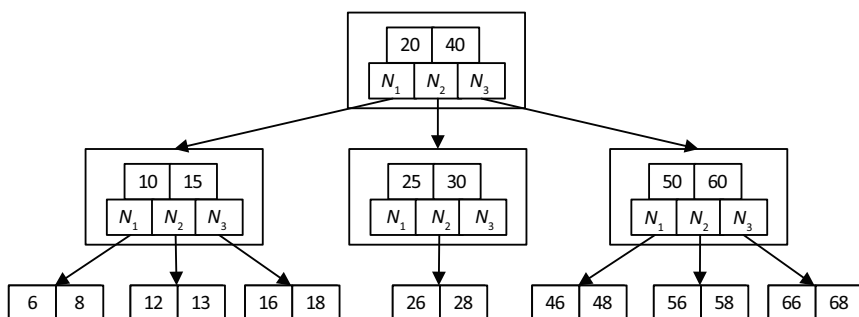


图 8-25 $m=3$ 的 B 树结构

B 树的特征包括：

- (1) 根节点的子节点数为 $[2, m]$ 。
- (2) 任意的非叶子节点最多有 m 个子节点，其中 m 大于 2。
- (3) 除根节点以外的非叶子节点的子节点数为 $[m/2, m]$ 。
- (4) 每个节点至少存储 $m/2-1$ （向上取整）和至多存储 $m-1$ 个信息。
- (5) 非叶子节点的存储信息个数等于子节点的个数减 1。
- (6) 对于非叶子节点的存储信息， $\text{Data}[1]$ 、 $\text{Data}[2]$ 、 \dots 、 $\text{Data}[m-1]$ ，其中， $\text{Data}[i] < \text{Data}[i+1]$ 。
- (7) 所有叶子节点均在同一层。
- (8) 非叶子结点的子节点 Nodes 集合中， $\text{Nodes}[i]$ 对应的所有 Data 值一定小于当前结点的 $\text{Data}[i]$ 。

B 树的查找方式与二叉排序树的查找方式类似，都是通过节点与子节点的大小关系进行查找，不同点在于 B 树在对存储节点进行值比较时，需要依次比较该节点中包含的所有值。找到相应值则表示找到，若没有找到，则再按照该节点的子节点递归查找，每个节点中的数据比较按照依次比较节点的所有值。

B 树的数据结构，对于磁盘 I/O 读 / 写有非常大的优化帮助，例如，假定

一个节点可以存储 100 个元素，那么三层的 B 树可以存储超过 100 万个元素信息。在工程应用中，常常采用 B^+ 树作为索引，而 B^+ 树是一种基于 B 树的模型。

8.3.3 B^+ 树

B^+ 树实质上是一种对 B^- 树的变异，应文件系统的需求而产生，它能够保证数据稳定有序，并且其修改和插入都是对数级别的时间复杂度，具有较高的处理性能，如图 8-26 所示。

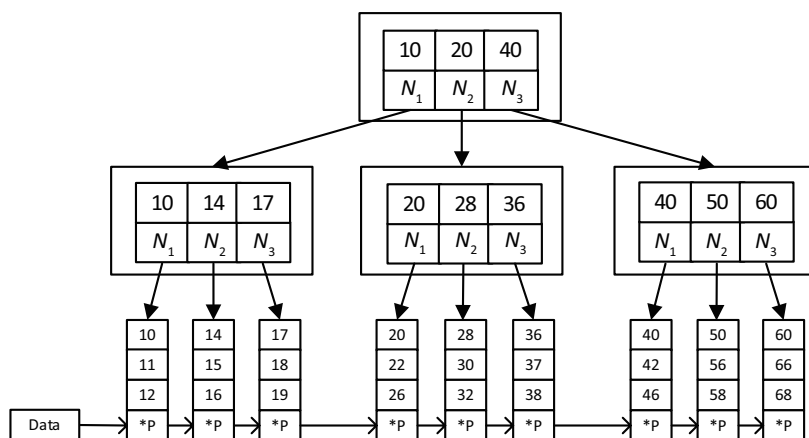


图 8-26 $m=3$ 的 B^+ 树的结构

除了同 B 树具有相同的特征之外，还包括：

- (1) 非叶子节点的子节点数量与存储信息数量相同。
- (2) 非叶子节点的子节点指针 $N[i]$ 指向存储信息属于当前节点的 $[Data[i], Data[i+1]]$ 的子树 (B^- 树是开区间)。
- (3) 所有存储信息都在叶子节点中出现。
- (4) 所有叶子节点中增加一个链指针。

B^+ 树的节点通常表示为一组有序的元素集合及子节点指针集。其查找方式类似于二叉搜索树。从根节点开始查找，然后从上往下依次查找，查找



过程中依然选择一定范围的子节点进行查找。此外，由于 B^+ 树的叶节点通过链表的方式关联起来，因此遍历所有数据的过程也可以通过叶节点的链表输出。

B^+ 树的插入过程：首先找到需要插入节点的位置，如果该节点中的数据没有达到饱满状态，将插入的值放置到该节点中即可。如果该节点已经达到饱满状态，则将该节点拆分为两个节点，所有值尽可能均匀分布在两个新节点中。在 B^+ 树中递归实现上述过程，直到处理到根节点，如果根节点被拆分为两个节点，则重新创建一个新的根节点。

B^+ 树的删除过程：首先查找到需要删除的值，由于 B^+ 树的特性，删除操作仅在叶子节点中进行。当删除的值是叶子节点中的最大值时，二叉搜索树、B 树及 B^+ 树，三者的区别在于：

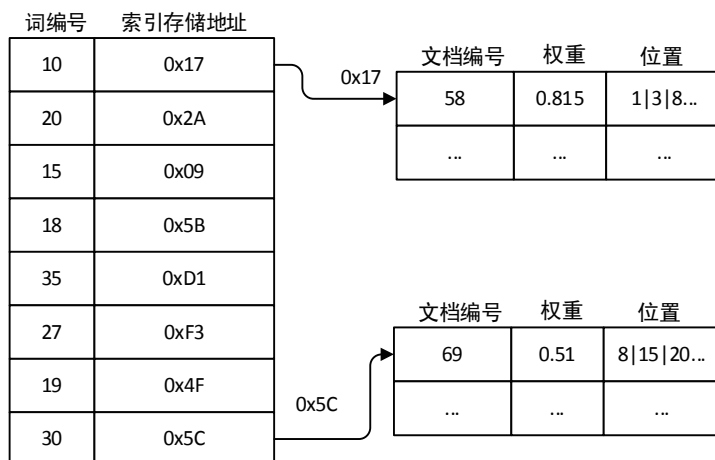
(1) 二叉搜索树为二叉树，每个节点仅存储一个信息，在查找过程中等于表示命中，小于则向节点的左节点继续查找，大于则向节点的右节点继续查找。

(2) B 树是一种多叉搜索查找树，每个节点存储 $m/2 \sim m$ 个信息，所有信息存储在整个树中，且仅出现一次，非叶子节点也可以在查找过程中命中。

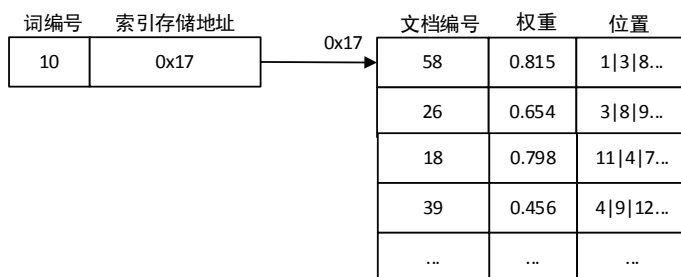
(3) B^+ 树是在 B 树基础上的一个变异，在所有叶子节点中添加了链表指针，所有信息存储在叶子节点中，而且在执行查找过程中，在叶子节点中查找结束，不会在非叶子节点中结束。

8.3.4 B^+ 树与文件索引

倒排索引中的文件数据以关键词作为“主键”按照 B^+ 树格式存储。给定的 B^+ 树中倒排索引数据如图 8-27 所示。

图 8-27 给定的 B⁺ 树中倒排索引数据

只需对词编号 10、20、15、18、35、27、19、30 建立 B⁺ 树索引即可。B⁺ 树的节点除存储词编号之外，还需存储索引存储地址。在这里实际上是通过 B⁺ 树建立一个词编号与对应存储的关系。不仅是利用 B⁺ 树构建一个这样的关系，还会在每次倒排索引数据插入过程中构建基于权重的 B⁺ 树，如图 8-28 所示。

图 8-28 利用 B⁺ 树作为倒排索引的文件索引

如图 8-28 所示，对于给定的一个词，需要按照权重构建索引，从高到底依次排序，也是通过 B⁺ 树构建，节点中除存储的权重之外，还会存储权重对应的文档编号。

这里之所以不采用哈希表存储这样的信息原因在于：



(1) 哈希表查找的时间复杂度 $O(1)$ 仅是一个数学上的期望值，在最坏的情况下，时间复杂度为 $O(n)$ ，不能避免表扫描情况。

(2) B^+ 树索引中不仅可以精确定位，还可以通过区域获取，例如获取词在文档中的权重大于某个值的所有文档。

(3) 哈希表在遇到大量哈希值相等的情况之后，性能不一定会比 B^+ 树的性能高。

(4) 在 B^+ 树中的数据扩展比较稳定，很容易实现数据的扩展，时间复杂度在可控范围之内。而哈希表在负载过大的情况下，性能波动明显，还会带来内存碎片，对于小型数据可以采用哈希表，但是对于数据量未知的数据采用哈希表存在一定风险。

总之，哈希表有其独特之处，但并不是所有场景都适合哈希表进行查询。 B^+ 树在文件索引中，更加适合，对保证系统稳定性有一定积极作用。

采用 B^+ 树构建索引而不采用 B 树构建索引的原因在于：

(1) 相同条件下， B^+ 树的磁盘 I/O 更低。 B^+ 树的内部节点中并不会存储具体与文件索引相关的信息，而是通过叶子节点一次性读取，所以 I/O 也会相对较低。

(2) B^+ 树的查询性能具有较高的稳定性。由于每一个具体信息均存储在叶子节点中，则所有查询的路径长度类似，则查询效率稳定，没有波动的情况。

8.4 字典树索引

字典树是一种非常常用的数据结构，又称为前缀树，是多叉树的一种，也是哈希树的变种。它与其他树的不同地方在于：结点的键值并不存储在该结点中，而是从根结点到该结点的字符串值相加，根结点默认是空字符串。

8.4.1 字典树索引概述

针对字典树存储“buy baby food”的英文单词，数据结构如图 8-29 所示。

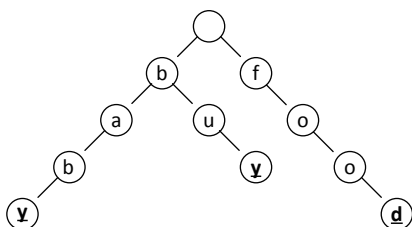


图 8-29 字典树结构

从图 8-29 可以看出：

- (1) 字典树的每个节点存储一个字母，相同父节点的子节点存在相同前缀。
- (2) 每个节点都有一个前缀，而叶子节点对应最长前缀。且叶子节点均为存储的单词。

因此，字典树的最大优点是利用字符串的公共前缀来减少存储大小及查询时间，拥有较高的查询性能。字典树的作用是主要用于搜索引擎的智能提示，如图 8-30 所示，输入“阿莫西林”能够有效地出现智能提示信息。

阿莫西林
阿莫西林胶囊
阿莫西林的作用
阿莫西林克拉维酸钾
阿莫西林过敏症状

图 8-30 智能提示

一个完整的搜索引擎，需要智能提示辅助用户进行搜索，以便于获得更好的搜索结果。字典树并不是一成不变，而是不断根据用户搜索关键词进行改变，包括排序上的变化。例如，当前月份智能提示如图 8-30 所示，但是在次月份，由于搜索“阿莫西林颗粒”的用户明显增多，并超越了“阿莫西林症状”，在

次月份智能提示中“阿莫西林颗粒”将会出现，排序会发生相应变化。

字典树的最大优点在于：利用字符串前缀来缩短查询时间和路径，极大地减少了不必要字符串的检索，并且减少了字符串的重复存储，查询效率高于哈希树。而智能提示主要是查找及插入操作，也非常符合字典树特性。

在字典树的使用过程中，也会将汉字转换为拼音进行存储，目的在于当用户搜索拼音时，也能进行智能提示。目前，常用汉字转拼音方法有两种：基于词典的方式和基于字符编码的方式。

（1）基于词典的方式。按照正常思路，将每个汉字与拼音建立一个哈希表，哈希表的键是该拼音，值是汉字对应的拼音列表。这样的方式简单可行，但从工程应用角度来看却不是很理想，原因在于不仅重复存储了相同的拼音，还需要较大的内存存储所有汉字集。

（2）基于字符编码的方式。国内用得最多的汉字编码是 GB 2312，它是中国国家标准总局制定的汉字编码字符集。GB 2312 编码中包含符号、数字、字母、日文、制表符等相关字符，其中最重要的组成部分是汉字。GB 2312 编码采用 16 位的编码方式，简体中文（Simplified Chinese）的编码范围从 B0A1 到 F7FE，如表 8-7 所示。

表 8-7 基于 GB 2312 编码的汉字示例

编码	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
B0A0		啊	阿	埃	挨	哎	唉	哀	皑	癌	蔼	矮	艾	碍	爱	隘
B0B0	鞍	氨	安	俺	按	暗	岸	胺	案	肮	昂	盎	凹	敖	熬	翱
B0C0	袄	傲	奥	懊	澳	芭	捌	扒	叭	吧	笆	八	疤	巴	拔	跋
B0D0	靶	把	耙	坝	霸	罢	爸	白	柏	百	摆	佰	败	拜	稗	斑
B0E0	班	搬	扳	般	颁	板	版	扮	拌	伴	瓣	半	办	绊	邦	帮
B0F0	梆	榜	膀	绑	棒	磅	蚌	镑	傍	谤	苞	胞	包	褒	剥	
...

通过表 8-7 可知，汉字“啊”的 GB 2312 编码为 B0A1，后面依次类

推，上述表不需要全部存储，通过 GB 2312 编码信息可以自动获得。在拥有 GB 2312 编码信息表之后，还需要一个所有汉语拼音列表，按照中华字典的顺序如下所示，并用数组存储。

"a", "ai", "an", "ang", "ao", "ba", "bai", "ban", "bang", "bao", "bu", "ca", "cai",
"can", "cang", "cao", "ce", "ceng", "cha", "chai", "chan", "chang", "chao", "che",
"chen", "cheng", "chi", "chong", "chou", "chu"...

通过建立 GB 2312 编码表与拼音列表的关系实现中文转拼音。

上述两种方法都有可取之处，也简单可行，能够实现单字的拼音转换，但是对于多音字两者都不能很好地匹配。为解决多音字问题，在上述实现基础上，还需要加入词典的方式。建立词与拼音的对应关系列表。使用词典意味着需要进行分词，给定一句话，首先通过分词，将每个词在拼音对应关系列表中去查找相应拼音，如果找到则结束，如果没有找到则拆分成字，按照基于字符编码或者字典查找的方式获得单字拼音。

同样，对于繁体中文与简体中文的相互转换，也需要通过繁体中文与简体中文词语的对应关系进行，不能按照单字的方式转换。繁体中文与简体中文并不是严格按照字进行转换，例如简体的“皇后”和繁体的“皇后”是一致的，而简体的“后来”对应的却是繁体的“後來”。简体中文的“后”字在不同语境下，转换方式不一样。

8.4.2 字典树索引构建

传统的字典树主要用于存储英文。实际上，字典树也可以存储中文，只是占用大小有变化。在英文环境中，字典树理论上共有 27 层，每层均有 26 个结点。实际上会远远小于这个数值，一个原因是英文字母的单词量并没有那么大，另一个原因是并不是所有节点都会有子节点。

对于中文而言，构建字典树有两种方式：一种方式是将中文拆解为单个词，分别存入字典树；另外一种方式是将中文转换为拼音，然后按照英文方式存储。在垂直领域搜索引擎中，一般只采用方法一。但是对于大多数搜索引擎，两种方法都会共同采用，原因是在用户搜索过程中，难免会以拼音作为输入。但是方法二中需要注意存在不同词语相同拼音的问题及拼音连续性问题。

(1) 不同词语相同拼音的问题。需要解决此问题，必须建立拼音对词语的对应关系。例如输入“kuaile”，返回词语中需要注意“快乐”及“快了”。

(2) 拼音连续性问题。在中文处理中，此类问题非常复杂，难于解决，一般依赖于上下文关系。例如输入“xian”，在直观情况下很难确定用户表达的是“现”还是“西安”。二者必须同时出现直到上下文出现。

两个问题只会在搜索引擎新上线过程中出现，因为当用户在输入“xian”之后，大部分用户均选择“西安”，通过反馈统计学习，在下一次用户输入“xian”时，出现“西安”即可。

字典树的搜索过程是从根节点开始的，将输入字符串进行拆解为字符数组，获取第一个字符，获取到根节点下该字符的子节点，然后依次进行第二个字符查找新的子节点，直至迭代到字符数组已经查找完毕，返回最后一个字符的子节点是单词的节点路径值即可，如同 8-31 所示，输入“ba”则会返回单词“ban”及“back”。

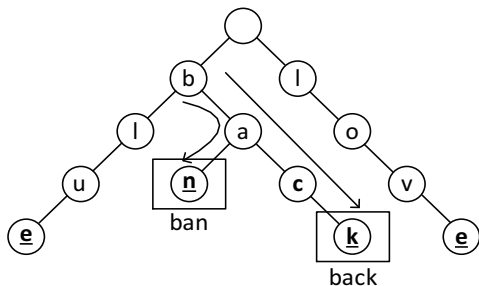


图 8-31 基于字典树的搜索示例

8.4.3 字典树查询优化

在字典树原始数据结构中，每层的查找均通过遍历方式进行，如图 8-32 所示，若需查询字符“f”是否在第二层节点中存在，只需遍历第二层节点即可，采用遍历的原因是英文只有 26 个字母，遍历查询性能比较可靠。但在实际应用过程中，尤其是在非英语地区文字处理中，这种方法往往会导致性能问题。以中文为例，每一层的节点可能数千。对于遍历一个比较长的中文字符串，查询性能将会明显下降。为了解决此问题，需要对字典树进行一次改进，如图 8-32 所示。

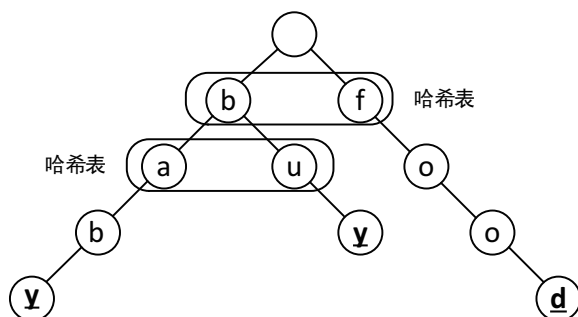


图 8-32 改进的字典树存储结构

改进后的字典树中，每个节点的键和节点指针将会被存入哈希表，利用哈希表可以快速查找的性能替代遍历的方式。通过更多空间换得更快查询性能。智能提示是搜索引擎非常重要的功能之一，且搜索频率非常高，因此智能提示的性能直接影响到用户对产品的体验，设计具备高性能查询的字典树非常有必要。

8.5 本章小结

构建索引作为继可扩展爬虫功能之后又一个非常重要的功能，作为搜索引



擎的神经中枢，通过构建高性能倒排索引和字典树索引，可以使搜索引擎检索服务达到更好的用户体验。倒排索引通过基于文档的分布式存储到各个服务器中，再利用 B^+ 树作为文件的存储索引并对索引数据进行排序、压缩处理。倒排索引和字典树作为搜索引擎的核心数据结构之一，改变了传统的数据查找方式，不仅在搜索引擎中使用，它们在各行各业都发挥着独特的魅力和优势。

第 9 章 搜索服务构建

搜索引擎通过网页或者移动应用对外提供搜索接口，搜索服务接口本身不具备搜索能力。搜索服务接口通过连接搜索服务后台，对外提供精准的搜索引擎服务。搜索服务后台在对数据检索过程中还包括对用户搜索词的分析，以及对搜索结果的排序、过滤、个性化分析等。

9.1 概述

本节将对搜索服务的体系结构和用户搜索问题的方法进行分析，然后通过一般搜索引擎的相关性排序讲解索引与搜索结果的关系，并对搜索中含有不安全信息过滤方式进行相关介绍。

9.1.1 体系结构

搜索引擎服务是将内部数据转换为信息价值的通道，以 WebService 或 WebSocket 等方式存在，整个体系结构如图 9-1 所示。

搜索服务器是整个在线搜索服务中非常核心的模块，关系到能否提供稳定、安全、可靠、及时的搜索服务。搜索服务器接收到搜索请求之后，首先通过缓存服务器进行查询，是否命中缓存，命中缓存的情况下则直接返回，反之再通过索引服务器获取索引信息，然后根据索引获得相应知识图谱信息及网页文档



信息，最终组合成搜索结果，返回给请求源。搜索服务器不仅仅是用户搜索的代理中转，还具备用户请求数据的初步分析能力。

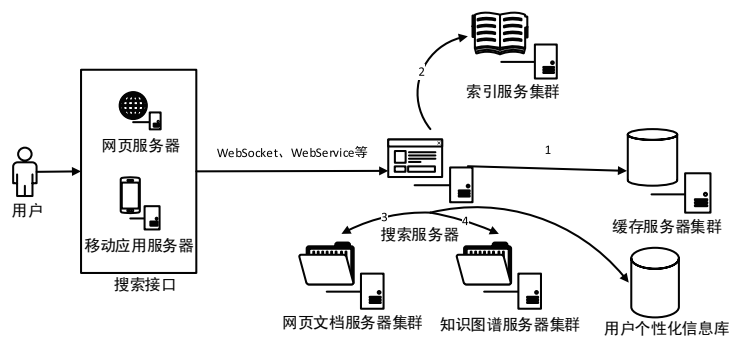


图 9-1 搜索接口与搜索服务示意

9.1.2 七何分析法

在当前搜索引擎中，通常会将用户的搜索提问归纳为“5W2H”，俗称七何分析法，简单且易于理解用户的意图，富有启发意义。广泛应用于人们的日常生活中，对系统决策和指向具有较好的辅助作用，可以有效掌控用户搜索的本质，完全抓住用户的搜索意图，使结果尽可能满足用户。

5W2H 是以 W 和 H 开头的英文单词，从搜索意图的角度去理解七何分析法，大致如表 9-1 所示。

表 9-1 七何分析法意图分析示意

5W2H	搜索意图	搜索示例
What	求搜索词的概念、含义、介绍	“股票是什么”“全息技术是什么”
How	求解决办法、处理过程	“蛋炒饭怎么做”“电脑蓝屏怎么办”
Why	求事情真相、原理性知识	“为什么北京这么堵”“人为什么会近视”
When	求发生时间	“什么时候阅兵式开始”
Where	求事物地理位置	“中关村在哪里”“哪里有电影院”
Who	求相关人物	“马云是谁”“谁是江南才子”
How Much	求程度、度量相关	“鲍鱼多少钱一个”“马云有多少财富”

一般情况下,用户搜索不会带有疑问词,例如搜索“全息技术”,表示“What”,意图搜索“全息技术是什么?”,返回结果带有“全息技术”的百科可能是比较好的答案。用户的搜索意图,基本上在 5W2H 的框架范围之内。

用户认知一个事情的过程总是这样:第一步,是什么;第二步,为什么;第三步,怎么办?例如,用户第一次听说“鱼鳞病”,那么了解过程应该是“鱼鳞病是什么”,其次是“为什么会得鱼鳞病”,最后是“得了鱼鳞病怎么办”,在这个过程中用户也会去了解与“鱼鳞病”相关的其他信息,但思路基本都是这样,如果在已经了解了第一步的情况下,用户会直接进行第二步或第三步搜索,例如“为什么马云那么富有”“怎么获得天使投资”,虽然不是所有的搜索都符合这个规律,但是却揭露了用户了解知识的一般性方式。

9.1.3 搜索语法

在搜索引擎的平常使用中,对大部分用户来说,很少用到高级搜索功能,但是高级搜索会给很多人士带来便利。因此,搜索引擎在检索过程中的语法设计也非常重要。一般有下面常用的 5 种高级搜索。

(1) 基于“intitle”的标题式搜索。搜索关键词仅限于在标题中搜索。标题一般是网站主题的表现,把搜索词限定在标题中,在某些时候能够获得较好的效果。搜索方式如“intitle: 马云”。

(2) 基于“inurl”的链接内容搜索。搜索关键词仅限于在链接中搜索。链接虽然在一般情况下没有实质信息,但是对于某些特定领域,针对 url 进行搜索会得到较好结果,例如搜索下载资源时,搜索方式如“inurl: .pdf”。

(3) 基于“site”的限定站点搜索。搜索结果仅限于指定一个或多个站点,例如,“搜索技术 site:sina.com”。

(4) 基于“filetype”的限定文件类型搜索。指定搜索结果的文件类型。网

上的媒体多种多样，有时候用户希望是特定媒体资源，例如 PPT 之类。例如，“搜索技术 filetype:ppt”。

(5) 基于双引号 (“ ”) 的文本包含搜索。通过对关键词进行双引号约束，则指定搜索结果返回的页面必须包含双引号中出现的所有词，包括顺序也需要一定匹配。例如搜索 ““马云阿里巴巴””。

上述基于双引号 (“ ”) 的文本包含搜索实质是布尔模型中的逻辑 “与” (×、AND) 操作，相似操作还包括逻辑 “或” (+、OR)、逻辑 “非” (—、NOT)，它们都是在布尔模型中对文档进行选择。

(1) 逻辑 “与”。在搜索过程中，通过 “与” 运算，则意味着需要将所有搜索词共同的文档筛选出来，在搜索引擎中逻辑 “与” 对于相关性搜索效果是最好的一种方式，但是输出的文档数量相对较少。例如搜索 “搜索技术”，通过分词得到 “搜索” 与 “技术” 两词。“搜索” 与 “技术” 通过倒排索引获得文档集合之后，将两者文档中的交集部分筛选出来。图 9-2 所示为逻辑 “与” 搜索 “搜索 AND 技术”。

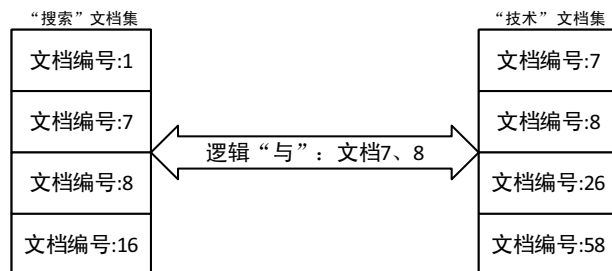


图 9-2 文档筛选过程中的逻辑 “与” 运算

(2) 逻辑 “或”。逻辑 “或” 在集合的概念中即为并集，是将所有关键词对应的文档求并集输出。默认情况下，搜索引擎采用逻辑 “或” 的方式处理搜索请求，不仅包含了逻辑 “与” 中的优秀结果，也提供了大量的参考文档。如图 9-3 所示为逻辑 “或” 搜索 “搜索 OR 技术”。

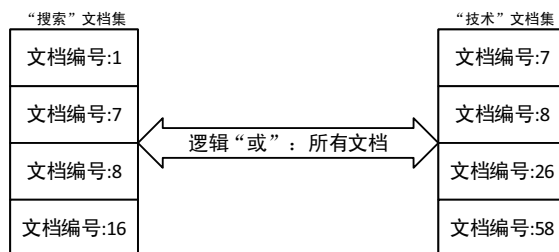


图 9-3 文档筛选过程中的逻辑“或”运算

(3) 逻辑“非”。逻辑“非”是指在搜索过程中,进行文档排除作用的搜索方式,在集合中表示集合相减。例如“搜索”与“技术”的逻辑“非”是指文档在关键词“搜索”的集合中,而不存在于“技术”的文档中。图 9-4 所示为逻辑“非”搜索“搜索 NOT 技术”。

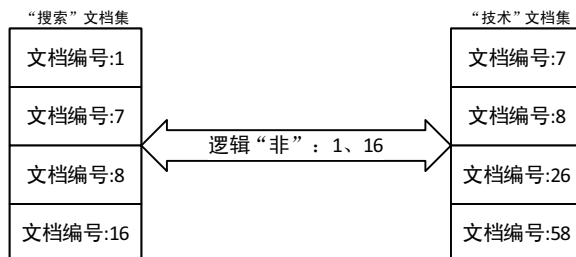


图 9-4 文档筛选过程中的逻辑“非”运算

在实际应用中,布尔逻辑运算可以混合使用,例如“兰州 AND 美食 NOT 兰州拉面”表示搜索兰州的美食,但是不包括兰州拉面。

9.1.4 相关性排序

相关性排序是依据用户的搜索词,根据倒排索引获得文档结果的排序方式。倒排索引的使用是搜索引擎服务中最为关键、重要的部分,它是相关性排序的依据。在相关搜索过程中,对于倒排索引的使用,与倒排索引的压缩也有关系。图 9-5 所示为通过词语哈希化及文档差值化后的倒排索引。



图 9-5 通过词语哈希化及文档差值化后的倒排索引

图 9-5 中的词语分别为“大数据”(564)、“搜索引擎”(986)。当用户搜索“大数据搜索引擎”时，会经过分词、索引获取、索引合并等相关过程。

(1) 词语分词和权重分析。将“大数据搜索引擎”分为“大数据”及“搜索引擎”，计算两者在搜索中的权重，词语在用户搜索词中的权重实质上是对用户搜索侧重点的分析。权重按照图 9-6 所示进行计算，首先对“大数据搜索引擎”进行分类分析，在确定属于类别“计算机 / 互联网”之后，通过查询“计算机 / 互联网”分类下已经离线计算好的所有词语的 TF-IDF 表，获得的值分别为 0.23、0.18，将两个值同比例扩大直到两者之和恰好为 1 时，最终分别为 0.56 和 0.44。

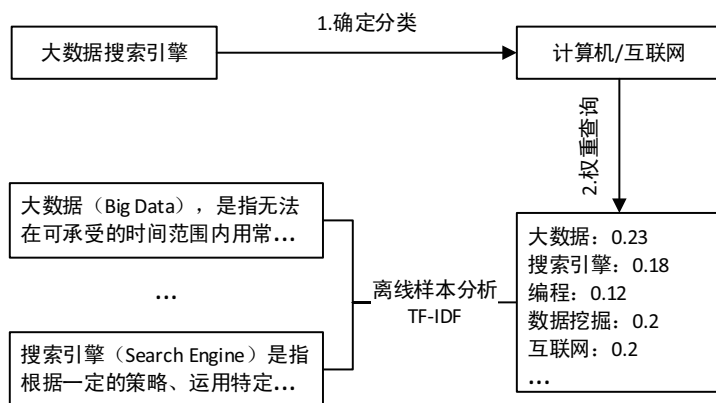


图 9-6 搜索词的权重获得

(2) 词语哈希化处理。“大数据”映射哈希值“564”，“搜索引擎”映射哈希值“986”，如表 9-2 所示。

表 9-2 词语“大数据”和“搜索引擎”哈希值与权重结果

词 语	哈希值	权 重
大数据	564	0.56
搜索引擎	986	0.44

(3) 索引查找。将词语进行一致性哈希算法，寻找在分布式索引存储中索引存储的具体位置，并读取相关数据，得到压缩后的倒排索引数据。

(4) 倒排索引解压。由于倒排索引中的文件编号是通过差值化压缩存储的，因此将两份数据进行解压，解压后如图 9-7 所示。



图 9-7 倒排索引解压后示意

(5) 索引文档权重计算。在索引构建过程中，每个关键字会根据在文档中的权重给定一个评分，表示这个词在这个文档中的重要性。在搜索过程中，也对关键词进行权重分析，表示用户搜索相关性的倾向，因此，需要把两者关键词权重进行结合，结合以后的评分如图 9-8 所示。

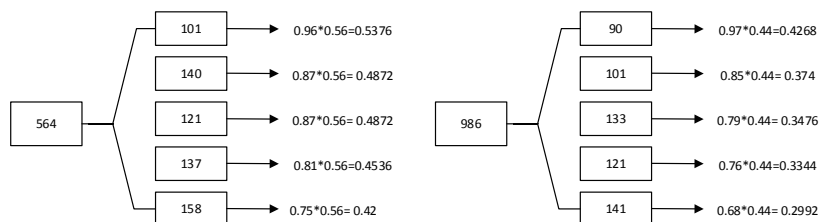


图 9-8 倒排索引中文档权重与关键词权重结合

(6) 文档合并，得到相关性排序结果。合并过程中，遇到相同文档则对两文档新的权值进行相加处理，最终所有文档按照词语的权值由高到底依次排序，如图 9-9 所示。

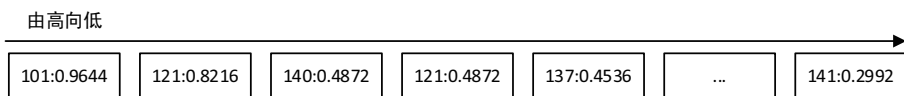


图 9-9 索引合并后的文档序列

因此，最终用户搜索“大数据搜索引擎”的相关性文档序列如图 9-9 所示。若存在更多词语的搜索则按照相同方式进行处理即可。值得说明的是，文档合并并不会成为性能瓶颈。原因在于索引构建过程中，会过滤掉词语权值较低的文档，不仅减少了相关性排序的计算，也节省了硬盘存储空间。

在上述过程的分词中，将“大数据搜索引擎”拆分为“大数据”与“搜索引擎”，在不同的搜索引擎中可能处理的粒度会不一样，分词粒度较细的可能是切分为“大”“数据”“搜索”“引擎”，细粒度的分词可以获得更多的文章信息，但是存在结果可能不够准确的情况，在对用户的搜索词进行细粒度分词时，可以按照图 9-10 所示进行。

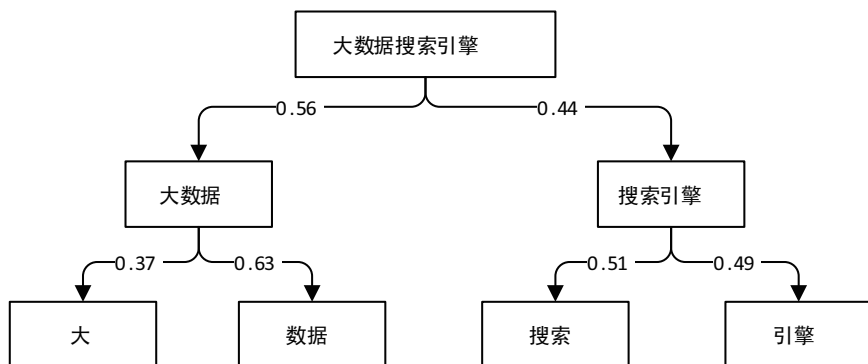


图 9-10 对用户的搜索词进行细粒度分词效果

图 9-10 中，不仅将“大数据搜索引擎”切分为“大数据”和“搜索引擎”，

还将分词后的词语再一次进行切分，并计算相应权重，虽然进行了二次切分，但并不是只对“大”、“数据”、“搜索”、“引擎”进行索引查找，“大数据”与“搜索引擎”也会存在，最终其细粒度分词及权重如表 9-3 所示。

表 9-3 “大数据搜索引擎”的细粒度分词及权重

词 语	哈希值	权 重
大数据	564	0.56
搜索引擎	986	0.44
大	178	$0.56 \times 0.37 = 0.2072$
数据	265	$0.56 \times 0.63 = 0.3528$
搜索	358	$0.44 \times 0.51 = 0.2244$
引擎	486	$0.44 \times 0.49 = 0.2156$

从表 9-3 中可以看出，伴随着分词粒度的细化，越细粒度的词语得到的权重越低。这样的权重分布可以兼顾搜索结果的准确性和广泛性。

相关性排序并不是搜索引擎的最终结果，仅仅是初步值，后续还会根据网页重要性评价、网页权威性评价、网页作弊评价等给予网页在排序上的调整。

9.1.5 不安全信息过滤

互联网中难免存在一些不安全信息，这些信息不仅敏感，而且对用户使用存在潜在风险，可能会遭遇木马病毒的危险。搜索引擎为保护网民的合法权益不被侵犯，几乎都会将此部分信息进行过滤。主要是通过两方面，一方面是通过不安全页面的过滤或者降低网页重要性和权威性，降低其在搜索结果中显示的可能性；另一方面从搜索入口防止用户主动进入不安全信息，主动通过搜索词进行屏蔽。Google 搜索引擎对不安全信息进行过滤如图 9-11 所示，当用户输入“成人电影网站”时，“成人电影”会被主动过滤，结果中只会对“网站”一词进行搜索。



图 9-11 Google 搜索引擎对不安全信息进行过滤

一般情况下，对不安全信息过滤可以首先将需要过滤的词汇放入哈希表；然后对用户的输入进行分词；最后将分词后的每个词在哈希表中进行查找，查找到则进行过滤。理论上是可行的，但是会存在如下两个问题：

(1) 伴随着需要过滤的词汇越来越多，哈希表越来越庞大。性能影响不仅受限于搜索词的个数，也与哈希表的大小相关。

(2) 分词并不能做到绝对的准确。例如，需要对词“成人电影”过滤，用户的输入为“成人电影频道”，如果分词为“成人电影”和“频道”则存在被过滤的可能性，但是若被分词为“成人”和“电影频道”则不会被过滤。

因此，采用哈希表的方式不能达到绝对的过滤效果，但是可以通过 Aho-Corasick 算法进行多模式匹配，解决上述问题。Aho-Corasick 算法是一种多模式字符串匹配算法，它通过将需要过滤的关键词建立字典树，从字典树的根节点依次向叶子节点进行跳转比较；如果比较成功则根据输出表确定被匹配的词语；如果比较失败，则根据失败表跳转继续匹配。因此，Aho-Corasick 算法需要确定跳转表、失败表和输出表。例如，需要过滤的关键词集合“语文”“学语文”“语言学”“语文教学”，对“大学语文教学”进行过滤分析示例。

(1) 构造跳转表。跳转表是一个模式匹配机，从数据结构上看实质是一个字典树。对“语文”“学语文”“语言学”“语文教学”依次构造字典树，以 0 表示字典树根节点。

第一步，将“语文”加入跳转表，如图 9-12 所示。

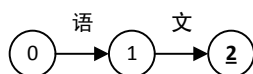


图 9-12 “语文” 构造跳转表

第二步，在上一步基础上，将“学语文”加入跳转表，如图 9-13 所示。

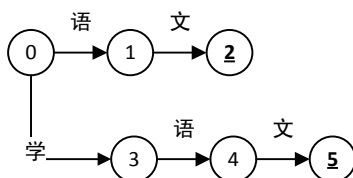


图 9-13 “语文” “学语文” 构造的跳转表

第三步，将“语言学”也按照相同的方法加入到跳转表中，如图 9-14 所示。

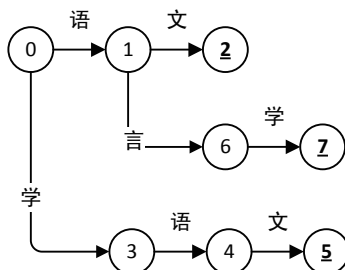


图 9-14 “语文” “学语文” “语言学” 构造的跳转表

第四步，将“语文教学”加入到跳转表中，如图 9-15 所示。

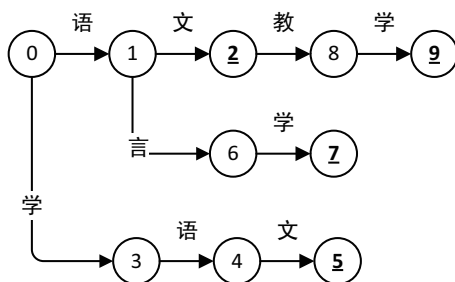


图 9-15 “语文” “学语文” “语言学” “语文教学” 构造的跳转表

通过上述四步，完成跳转表的构造。

(2) 构造失败表。失败表是在跳转表中对词语匹配失败之后，对下一步跳转的路径表示方法，是对字典树的扩充表达。规定字典树中深度为 1 的节点，失败后均跳转到根节点 0，节点 1 和节点 7 匹配失败后，下一步跳转到根节点、0，函数表达即为 $f(1)=0$ 、 $f(3)=0$ ，依次计算出 $f(1)\sim f(10)$ ，如图 9-16 所示，虚线为失败后跳转后路径， $f(1)=0$ 、 $f(2)=0$ 、 $f(3)=0$ 、 $f(4)=1$ 、 $f(5)=2$ 、 $f(6)=0$ 、 $f(7)=3$ 、 $f(8)=0$ 、 $f(9)=3$ 。

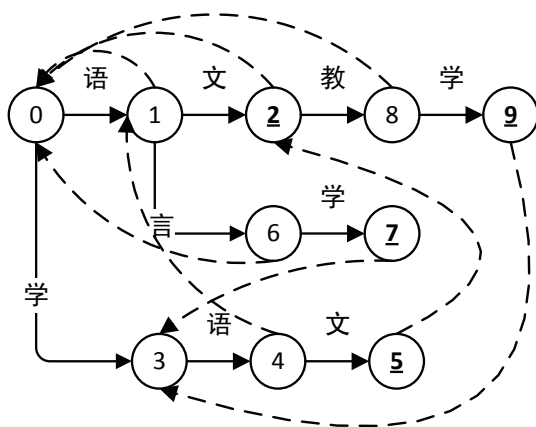


图 9-16 “语文”“学语文”“语言学”“语文教学”构造的失败表

(3) 构造输出表。输出表是表示在某个节点可以输出的词语集合，集合为 $\text{Output}(2)=\{\text{语文}\}$ 、 $\text{Output}(5)=\{\text{学语文, 语文}\}$ 、 $\text{Output}(7)=\{\text{语言学}\}$ 、 $\text{Output}(9)=\{\text{语文教学}\}$ 。

构造完跳转表、失败表及输出表之后，即可进行关键词过滤。例如，对输入“大学语文教学”，通过对其中的每个字进行跳转表匹配，如果字没有合理匹配到下一步成功跳转的对象，则按照失败表进行回退跳转，如表 9-4 所示。

表 9-4 “大学语文教学”的匹配过程

字	状态变化	输出信息
大	0->0	—
学	0->3	—
语	3->4	
文	4->5	学语文、语文
教	5->2->8	
学	8->9	语文教学

根据状态分别计算在状态 5、9 时的输出，而 $\text{Output}(5)=\{\text{学语文，语文}\}$ 、 $\text{Output}(9)=\{\text{语文教学}\}$ 。因此，在输入“大学语文教学”时，会查找到关键词“学语文”“语文”“语文教学”。

通过上述分析可知，Aho-Corasick 算法在匹配文本时，不会对文本进行回溯，因此，时间复杂度为 $O(n)$ ，时间复杂度与被过滤的关键词数量及长度无关，是一个性能极佳的多模式匹配算法。

9.2 大数据分布式缓存

在搜索引擎中，考虑到少数搜索词占据较大的搜索流量，即大部分用户都在搜索相同或者相似的信息，以及避免每次搜索结果均通过磁盘获取，提升搜索性能，大数据分布缓存系统在搜索引擎中功不可没。

9.2.1 缓存结构设计

在 8.1.5 节中介绍了分布式索引存储机制，分布式缓存同分布式索引存储机制类似。利用环形哈希空间和虚拟节点解决分布式缓存问题。设计结构如图 9-17 所示。两者不同点在于，分布式缓存的数据在缓存存储服务器中以内存方式记录，而分布式搜索存储中，倒排索引文件采用的是硬盘存储。

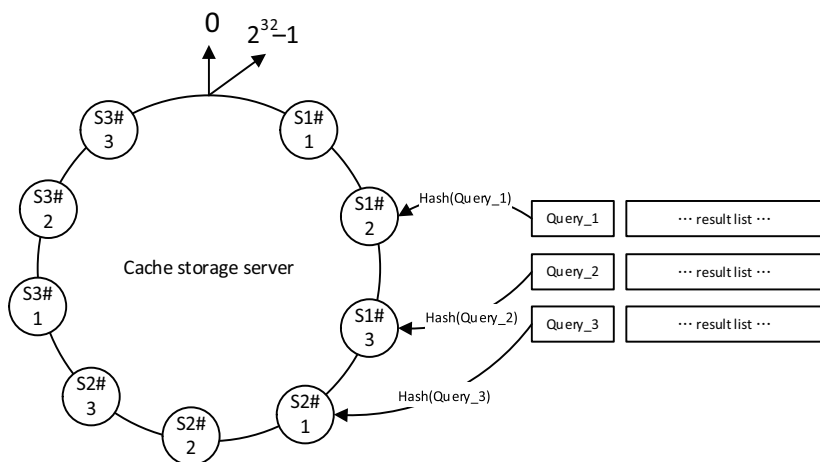


图 9-17 分布式索引存储机制设计结构

9.2.2 缓存更新策略

如果把整个分布式缓存视为一个整体，它将是一个队列，缓存的更新即为队列的更新问题。常常使用最不经常使用（Least Frequently Used, LFU）、先进先出（First In First Out, FIFO）、最近最少使用（Least Recently Used, LRU）三种缓存策略。

（1）最不经常使用 LFU。LFU 算法是将指定一段时间内被访问次数最少的搜索缓存替换出去。但是基于使用频率的数据淘汰机制 LFU，会导致进行频率统计，计算过于复杂，对于缓存系统会有性能影响。

（2）先进先出 FIFO。而传统的数据淘汰算法 FIFO 虽然操作简单，但是不能达到缓存的目的，几乎每次搜索都会有数据淘汰。

（3）最近最少使用 LRU。LRU 算法基于：最前面被频繁使用的搜索词中，在后面很有可能被再次使用。反之，越久未被使用的搜索词，后面使用的可能性越低。LRU 算法设计比较合理，工程中实现起来相对比较复杂，系统开销也相对较大，但是搜索引擎的工程实践表明，采用 LRU 算法的缓存命中率极高。

LRU 算法与 LFU 算法的区别在于，LRU 淘汰的是最长时间未被再次使用的搜索词，LFU 淘汰的是一定时间内被访问次数最少的搜索词，因此选择 LRU 算法比较合适。

除数据淘汰之外，对于分布式缓存，还有非常重要的一项，即数据同步，例如，最近一个星期用户对“全面开放二孩政策”搜索的流量比较高，因此一直处于分布式缓存中。但是一个星期后的今天，再次搜索“全面开放二孩政策”，由于后台索引和数据已经发生较大变化，倘若依然从缓存中取数据，将会导致信息不及时。因此，将及时的信息对象进行同步，对于搜索引擎这类数据经常变化的系统来说尤为重要。

对于缓存对象同步，可以采用数据触发机制，在分布式缓存中，给定每个缓存搜索词的最近一次缓存时间，当用户搜索请求到达分布式缓存一层时，不仅检查缓存是否存在，还查询缓存是否在有效期之内，如果存在，但是不在有效期内，则不从缓存系统中获取数据，从搜索代理服务器中获取搜索结果，同时需要将新的搜索结果同步到相应的缓存中，并更改缓存时间。

9.3 文本纠错算法

用户在搜索时，难免存在将搜索词输错的情况，即使是在智能提示的作用下也存在输错的情况。因此，能够智能进行文本纠错也是对提示搜索效果非常重要的一环。文本纠错包含中文文本纠错和英文文本纠错。

9.3.1 中文文本纠错

例如，当用户试图搜索“彩虹桥”，但是却输入成“彩红桥”。矫正用户的错误搜索关键词，需要从判定和纠正两方面进行解决。整个过程可以采用

N-Gram 语言模型对中文文本纠错。

N-Gram 语言模型基于这样一个假设：第 n 个字的出现仅仅只与前面第 $n-1$ 字相关。倘若一个搜索词 Q 包含 n 个字，即 $Q = \{w_1, w_2, w_3, \dots, w_n\}$ ，则判定这个搜索词是否合理，可以转换为如下形式：

$$P(q)=P(w_1,w_2,w_3,\cdots,w_n)=P(w_1)*P(w_2 \mid w_1)*\cdots*P(w_n \mid w_1,w_2,\cdots,w_{n-1})$$

$P(q)$ 即为搜索词 Q 是否合理的概率。但是倘若 n 值过大，则计算前面 n 个词时会造成矩阵过于稀疏。因此，简化 N-Gram 语言模型至 Bi-Gram 模型——二元语言模型，即一个字 w_i 的出现仅依赖于它的前一个词 w_{i-1} 。公式可简化为：

$$P(q)=P(w_1,w_2,w_3,\cdots,w_n)=P(w_1)*P(w_2 \mid w_1)*P(w_3 \mid w_2)*\cdots*P(w_n \mid w_{n-1})$$

而

$$P(w_i|w_{i-1}) = \frac{\text{total}(w_i|w_{i-1})}{\text{total}(w_{i-1})}$$

其中， $\text{total}(w_{i-1})$ 表示字 w_{i-1} 出现的总次数。 $\text{total}(w_i|w_{i-1})$ 表示 w_{i-1} 与 w_i 相邻出现的总次数。 $\text{total}(w_{i-1})$ 与 $\text{total}(w_i|w_{i-1})$ 均是通过语料库训练而来。这里的语料库不是人工收集，而是所有用户日积月累的搜索词，利用用户积累的搜索词可以发现新词汇及网络词汇，例如“云备胎”。

因此，基于上述模型，可以针对部分数据集作为示例验证“彩虹桥”是一个错误的搜索词。通过语料库统计出的 $\text{total}(w_{i-1})$ 如表 9-5 所示。

表 9-5 “彩”“红”“桥”语料库中出现的总次数统计

彩	红	桥
9 620	3 658	5 564

$\text{total}(w_i|w_{i-1})$ 如表 9-6 所示。

表 9-6 “彩”“红”“桥”语料库中出现的相邻次数统计

	彩	红	桥
彩	162	99	269
红	3	152	23
桥	2	1	167

通过 $\text{total}(w_{i-1})$ 、 $\text{total}(w_i|w_{i-1})$ 计算 $P(w_i|w_{i-1})$ ，如表 9-7 所示。

表 9-7 “彩”“红”“桥”语料库中出现的相邻概率统计

	彩	红	桥
彩	$162/9\ 620 \approx 0.017$	$99/9\ 620 \approx 0.011$	$269/9\ 620 \approx 0.028$
红	$3/3\ 658 \approx 0.001$	$452/3\ 658 \approx 0.124$	$23/3\ 658 \approx 0.006$
桥	$2/5\ 564 \approx 0.0$	$1/5\ 564 \approx 0$	$362/5\ 564 \approx 0.065$

因此，综上计算 $P(q) = 9\ 620 / (9\ 620 + 3\ 658 + 5\ 564) * 0.011 * 0.006 \approx 0.000\ 034$

计算出的值虽然非常小，但还不能判定它不是一个无误的搜索词。在语料库训练时还需要训练出一些值，即搜索词可能性最小阈值 k_i ， i 表示搜索词的字符个数，不同字符个数的搜索词 k 值也不一致，工程中 i 值最大设定为 255。如果 $P(q) < k_i$ 则可以判定是有误的搜索词。

在完成验证搜索词“彩虹桥”是一个有误的搜索词之后，还需要实现纠错功能。一种方法是“彩虹桥”转换为拼音“caihongqiao”。在智能提示中匹配第一个推荐的拼音为“caihongqiao”的搜索词。此类方法只能作为一种辅助方法，因为存在这样的可能性：用户的输入词不仅字错了，连音也错了，例如试图搜索“神魂颠倒”，却输入成“绳魂颠倒”，此类错误采用拼音的方式，不能有效完成纠错。

笔者曾经尝试用词库的方式纠正错误的字，由于不确定是哪个字出错，导致计算的复杂度较高，但采用最短编辑距离（Edit Distance）是一种比较可行的方式，计算距离意味着字符串两两之间进行比较，是否对性能会造成影响，搜索引擎后台有数十万不同的热搜词，一一比较不太可取，例如，当前是对“绳



魂颠倒”这个词纠错，则只会取出四个字的热搜词进行比较，相对来说比较次数会减少很多。

最短编辑距离的思想是指将一个词语改变为另外一个词语付出的代价，若代价越小，则说明越相似。算法思路大致如下：

(1) 设定输入字符串 $S=s_1s_2\cdots s_m$, $T=t_1t_2\cdots t_n$ 。

(2) 构建 S 和 T 的 $(m+1)*(n+1)$ 阶关系匹配矩阵，默认矩阵第一列为 S ，第一行为 T 。 $d_{i,j}$ 表示矩阵中第 i 行第 j 列的值。

(3) 按照如下过程填充 $(m+1)*(n+1)$ 阶关系匹配矩阵。

$$d_{i,j} = \begin{cases} i & j=0 \\ j & i=0 \\ \min(d_{i-1,j-1}, d_{i-1,j}, d_{i,j-1}) + a_{i,j} & i,j>0 \end{cases}$$

其中，

$$a_{i,j} = \begin{cases} 0 & s_i=t_j \\ 1 & s_i \neq t_j \end{cases} \quad (i=1,2,\cdots,m; j=1,2,\cdots,n)$$

$\min(d_{i-1,j-1}, d_{i-1,j}, d_{i,j-1}) + a_{i,j}$ 的实际含义指如果两个字符相等，则以此位置为基础，在其左上角、上、左三个位置中取出最小的值；若不相等则在此位置的左上角、上、左三个位置中取出最小的值后再加上 1。

(4) 确定编辑距离。矩阵中 $d_{m,n}$ 的值即为 S 和 T 的编辑距离。以“绳魂颠倒”“神魂颠倒”计算编辑距离，如表 9-8 所示。

表 9-8 “绳魂颠倒”“神魂颠倒”的编辑距离计算

		神	魂	颠	倒
	0	1	2	3	4
绳	1	1	2	3	4
魂	2	2	1	2	3
颠	3	3	2	1	2
倒	4	4	3	2	1

因此，通过表 9-8 计算，“绳魂颠倒”“神魂颠倒”的编辑距离为 1，字符串相似度为 $1-1/4=0.75$ 。

同理可以计算“彩红桥”与“彩虹桥”的编辑距离，如表 9-9 所示。

表 9-9 “彩红桥”与“彩虹桥”的编辑距离计算

		彩	虹	桥
	0	1	2	3
彩	1	0	1	2
红	2	1	1	2
桥	3	2	2	1

通过表 9-9 的计算可知，“彩红桥”与“彩虹桥”的编辑距离为 1，字符串相似度为 $1-1/3 \approx 0.67$ 。

综上所述，已经可以通过编辑距离计算出与有误搜索词相似的搜索热词，但是不能忽略其搜索热度对纠正的影响，当编辑距离一致时，以搜索词的热度作为排序依据。

值得注意的是，为及时发现新词，避免把新词当作错误词汇。语料库需要根据实际用户量，决定每天或者每周进行一次更新，这也是智能搜索的一部分，倘若将正确的词进行错误判定，将会大大减少用户好感。例如，网络热词“不造”，当用户输入搜索词后，倘若没有及时发现这类网络热词，很容易提示纠正为“不燥”。

9.3.2 英文文本纠错

在用户输入过程中，不仅有中文，还包括英文，英文拼写纠错与中文纠错一样显得非常重要。英文中的拼写错误包括单词拼写错误和单词输入错误两种情况。

(1) 单词拼写错误。是常规错误，表示单词本身就不存在，例如，将“Gift”输入成“Gifft”，是不小心多输入字母“f”导致错误。

根据以往用户搜索历史记录统计，在用户的英文单词拼写错误中，超过一



半是单词错误。而单词错误中，大部分是输错一个字母，几乎所有的错误都没有输错超过两个字母的情况。因此，通过字典匹配出错误的单词与字典中单词编辑距离小于等于 2 的所有单词即可。例如，用户输入关于单词 A 和 B 的搜索，分别记作 w_A 、 w_B ，发现 w_B 为单词错误，通过文本编辑距离计算，得到与 w_B 的编辑距离不超过 2 的词： $w_{B,1}$ ， $w_{B,2}$ ， \cdots ， $w_{B,n}$ ，需要确定最佳建议单词，需要计算 $P(w_A|w_{B,i})$ ，即 w_A 出现在 w_B 的候选词 $w_{B,1}$ ， $w_{B,2}$ ， \cdots ， $w_{B,n}$ 前面的概率，将最大的概率确定为最佳建议。

(2) 单词输入错误。用户输入过程中的确输错词，但是输错的词依然是一个合法的单词，例如，将“riot”输入成“rive”，两者都是英语词典中存在的单词，但是意义却完全不一样。非单词错误在英文中属于难度较大的错误纠错，因为句子中输入的每个单词都有可能是错误的。因此，解决此问题，需要关联文字的上下文，对于一个给定输入的句子： $S = \{w_1, w_2, w_3, \cdots, w_n\}$ ，需要为里面的每个单词确定其可能的候选词，候选词通过字符串相似度获得，如表 9-10 所示。

表 9-10 单词与候选词的对应关系

输入词	候选词
w_1	$w_{1,1}, w_{1,2}, w_{1,3}, \cdots$
w_2	$w_{2,1}, w_{2,2}, w_{2,3}, \cdots$
\cdots	\cdots
w_n	$w_{n,1}, w_{n,2}, w_{n,3}, \cdots$

通过表 9-10 可知，已经将问题实质转换为候选词之间的组合概率问题，可通过维特比算法直接计算，与中文的处理方式极其类似。

9.4 结果显示算法

结果显示在搜索行为触发后，搜索系统为用户行为做出的展示性行为，站内内容包括网页标题、链接、动态摘要等，甚至还包括网页快照信息。

9.4.1 动态摘要

在搜索结果中，摘要内容会根据搜索关键词的不同位置生产动态摘要，动态摘要的价值在于帮助用户在未点击该链接的情况下，确定该链接是否符合自己的预期。如果动态摘要提取不合理将会导致下列两种情况：

(1) 网页与用户的搜索词相关，但是用户发现摘要不是想要的内容，导致用户忽略该链接。

(2) 网页与用户的搜索词不相关，但是用户发现摘要看起来像是想要的内容，导致用户误点击此链接。

因此，在动态选取摘要时，应当具备如下两个原则：

(1) 最大化覆盖到用户搜索词的摘要。

(2) 最大化体现网页正文表现的内容。

在优先满足第一条的情况下，尽可能满足第二条原则。之所以将第一条优先考虑，是考虑到用户的搜索词不一定是网页最想表达的内容，可能是其中一点局部信息，但从用户角度来讲，这部分局部信息是有价值的。在动态摘要中，还需要保证摘要的完整性、丰富性和可理解性。

基于上述对动态摘要的要求，在对用户的搜索词进行分词之后，在搜索结果的正文内容中，定位词汇在正文中的内容，以关键词为中心寻找前后分句位置，形成摘要。例如，用户搜索“Java 高级编程”之后，每篇文章需要取前约 50 字摘要，搜索结果中某文章有正文如下所示：

北京“优秀程序员”训练营 2016 年开始啦，精选各类教程。《Java 指南》探讨了 Java 技术的高级特性，包括许多与 Java 语言相关的开源技术。全书共 14 章，前 3 章介绍了高效 Java 开发人员具备的基本素质。

通过对正文分词,并在分词后的结果中标记出词汇“Java”、“高级”、“编程”,如下所示:

北京“优秀程序员”训练营 2016 年开始啦,精选各类教程。《Java 指南》探讨了 Java 技术的高级特性,包括许多与 Java 语言相关的开源技术。全书共 14 章,前 3 章介绍了高效 Java 编程人员具备的基本素质。。

一种最简单的方式是通过 50 字滑动窗口模型,以搜索词作为滑动窗口开始位置,依次向后推送窗口,记录在 50 字内,滑动窗口中选择搜索词包含最多的句子,并截取成摘要。滑动窗口的方式易于理解和实现,但是对于满足完整性和可读性较差,而且忽视了摘要不一定是连续的句子,采用此类滑动窗口的方式不可取。可以换一个角度,从摘要的完整性出发,人们阅读一篇文章或者一段话,都是习惯性地从句子的起始位置进行阅读。因此,可以将正文摘要拆分为几句话,按照句子与搜索词的相关性进行分析,然后将所有句子的相关性从高到低进行排序,从第一句话开始取摘要,然后依次取,直到取满 50 字为止,需要注意的是,相关性的第二句有可能在正文中是相关性第一句的前面,因此取的时候,不仅仅需要按照相关性,还需按照句子在正文中的前后逻辑依次取。

继续上述示例,计算“Java”“高级”“编程”在各句子中的重要性,如表 9-11 所示。

表 9-11 计算“Java”“高级”“编程”在句子中的重要性

句子编号	内 容	搜索词与其重要性
1	北京“优秀程序员”训练营 2016 年开始啦,精选各类教程	0
2	《Java 指南》探讨了 Java 技术的高级特性,包括许多与 Java 语言相关的开源技术	0.472 015
3	全书共 14 章,前 3 章介绍了高效 Java 编程人员具备的基本素质	0.386 549

表 9-11 中,计算了搜索词在各个句子中的重要性,这个重要性可参考 3.6 节进行关键词权重分析。以“《Java 指南》探讨了 Java 技术的高级特性,包括许多

与 Java 语言相关的开源技术。”为例，则是计算搜索词“Java”“高级”的权重之和，若采用 TF-IDF 算法，需要将其中的逆向文档频率（IDF）提前在数据均匀的语料库中训练，应用时只需计算各个搜索词在各句子中的频率（TF）即可。

针对表 9-11，对句子按照重要性排序，句子编号则为“2 3 1”，先取第二句，然后取第三句，每次取完验证长度，因此示例的摘要为：

《Java 指南》探讨了 Java 技术的高级特性，包括许多与 Java 语言相关的开源技术。全书共 14 章，前 3 章介绍了高效 Java 编程…

如果示例中最终相似度排序是“2 1 3”，则也只能取第二句和第三句，后面取的句子编号一定比当前取的句子编号大，保证摘要正常语序，如果取得的句子编号之间不连续，在摘要中需要通过“…”表示中间有跳过。之所以会把搜索词在正文的相对位置作为摘要权重的依据，是因为正文中搜索词的相对位置越近，则搜索结果的相关性越好。

将正文的内容进行句子拆分保证了完整性，具备一定的可读性，对每个句子进行重要性排序，然后取摘要保证了内容的丰富性，以及具备很好的参考阅读价值，保持摘要中句子前后之间的关系，保留了摘要的可理解性，还利用位置关系过滤掉不完整的句子。最终保证摘要满足一定要求。

当搜索结果中包含网站首页时，一般情况下，使用静态摘要显示，例如，搜索“新浪”，返回结果是 <http://www.sina.com>，则无须对内容进行动态摘要分析，一方面对于非正式内容的页面，进行动态摘要提取效果不好；另一方面，新浪首页本身也提供了相应的静态摘要，对于新浪的静态摘要如下所示，直接截取前 50 字作为摘要即可。

新浪网为全球用户 24 小时提供全面、及时的中文资讯，内容覆盖国内外突发事件、体坛赛事、娱乐时尚、产业资讯、实用信息等，设有新闻、体

育、娱乐、财经、科技、房产、汽车等 30 多个内容频道，同时开设博客、视频、论坛等自由互动交流空间。

9.4.2 关键词高亮算法

关键词高亮是为提供更好的搜索体验，将用户的搜索词在文本中进行高亮显示。虽然在索引构建过程中，可以考虑通过存储关键词在文档中的位置，从而将该位置的关键词高亮。但是由于在索引中存储大量的位置信息，会导致大量的硬盘空间存储，因此，部分搜索引擎在构建索引过程中仅仅利用关键词位置计算权重，但是并不会存储关键词在文档中的位置信息。例如，设定句子为“猫鼠各一物，一物降一物”，以及搜索词“一物降一物”，获取搜索词在句子中的位置。

在字符串位置查找算法中，Knuth-Morris-Pratt 算法（KMP 算法）是一种比较高效的查找方式。KMP 算法通过词在不能正确匹配时记录下的信息确定下一个匹配点的位置，而不是从头开始重新比较，从而达到避免字符串重复比较的方法。步骤方法如下：

（1）字符串比较。将句子与关键字分别以字符串数组的方式，从字符串的 0 位置开始进行比较，如图 9-18 所示。

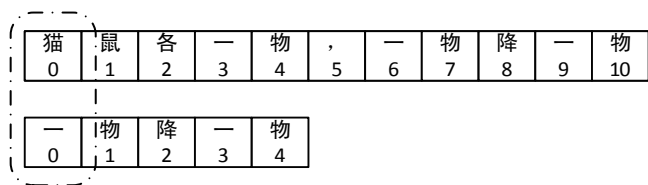


图 9-18 两个字符串头部开始比较

由于“猫”和“一”不一样，将被匹配的文本字符串位置后移一位，如图 9-19 所示。

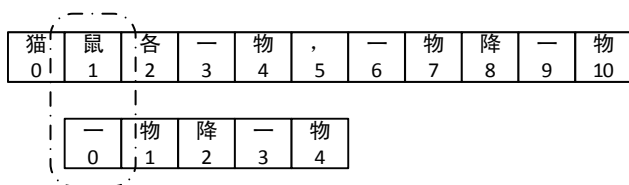


图 9-19 被匹配字符串下移一位比较

(2) 依次对被匹配字符串中的文本进行比较, 直到被匹配字符串的第 3 位置与关键词的第 0 位置匹配, 如图 9-20 所示。



图 9-20 字符串匹配头部字符

(3) 如图 9-20 所示, 被匹配字符串与关键词分别后移一位, 即被匹配字符串的第 4 位置与关键词的第 1 位置比较, 如图 9-21 所示。

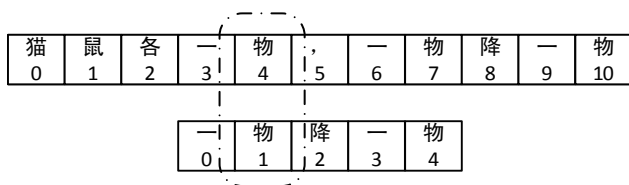


图 9-21 字符串匹配后同时后移一位

(4) 在同时后移一位之后, 发现两者在该位置上的字符一致, 因此继续同时向后移动一位, 如图 9-22 所示。

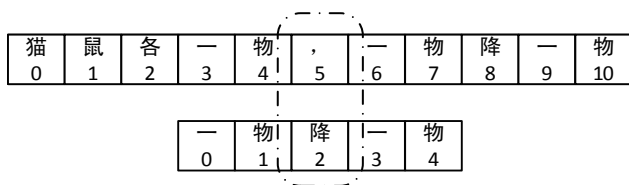


图 9-22 字符串后移一位之后, 字符不相同



(5) 如图 9-22 所示，在被匹配字符串的第 5 个位置与关键词的第 2 个位置，两者字符并不相同，这意味着从被匹配字符串的第 3 个位置开始匹配失败。按照一般的字符串比较算法，这时需要将被比较字符串的比较位置从 5 移到 4，将关键词的位置从 2 移到 0，如图 9-23 所示。

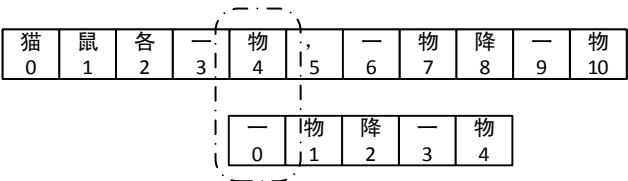


图 9-23 字符串重新归位重新比较

根据图 9-23，如果将被比较字符串的第 4 个位置与关键词的第 1 个位置进行比较，这样的方法是可行的，但是实质被比较字符串的第 4 个位置在上一次中已经被比较过，再次进行比较性能上有一定影响。KMP 算法在这个步骤中，做了一定优化处理。利用部分匹配表，在此处进行优化跳转，向前移动位数等于关键词已经匹配的位数与部分匹配表对应位置的差。

部分匹配表根据关键词的前缀及后缀的共有元素的最长长度，关键词需要拆分为相应的匹配可能子集，“一物降一物”的部分匹配表计算如表 9-12 所示。

表 9-12 部分匹配表计算

匹配子集	前缀	后缀	共有元素 最长长度
一	{null}	{null}	0
一物	{“一”}	{“物”}	0
一物降	{“一”，“一物”}	{“降”，“物降”}	0
一物降一	{“一”，“一物”，“一物降”}	{“一”，“降一”，“物降一”}	0
一物降一物	{“一”，“一物”，“一物降”}	{“物”，“一物”，“降一物”，“物降一物”}	2

(6) 在图 9-23 所示位置中，由于匹配字符串的第 5 个位置与关键词的第 2 个位置字符不匹配，将关键词的位置向前移动，移动位数为关键词已经匹配的

位数 2 与已经匹配的部分关键词“一物”对应的共有元素最长长度 0 之差，即向前移动位数为 2，对应关键词的第 0 位置。移动后如图 9-24 所示。

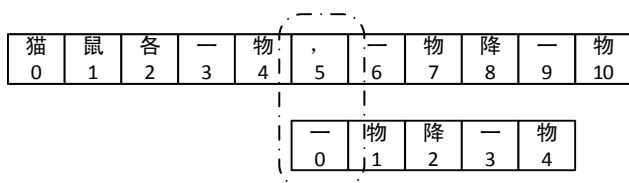


图 9-24 匹配失败后根据部分匹配表调整关键词匹配位置

(7) 在通过部分匹配表进行调整关键词之后，依然无法匹配上，则被匹配的字符串后移一位依次进行比较，不断重复上述过程，最终匹配到关键词，如图 9-25 所示。

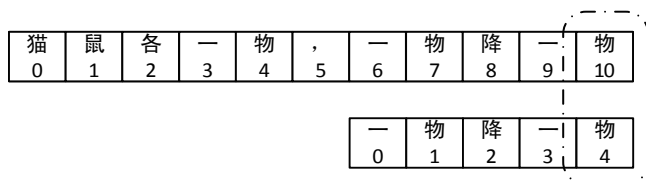


图 9-25 字符串与关键词成功匹配

经过上述过程，可以发现对于被匹配的字符串不存在重复匹配的问题，都会依次往下进行匹配，匹配性能会高于字符串一一对照比较的方式。唯一不同的是，需要将关键词建立部分匹配表，建立部分匹配表的时间复杂度为 $O(n)$ 。匹配表的实质是字符串中难免存在重复的情况，例如，字符串“土豆不是土豆”中存在两个“土豆”，那么“土豆”的部分匹配值为 2，在关键词移动时，第一个“土豆”向后移动 4 位（字符串长度 6 与部分匹配值 2 的差），就到达第二个“土豆”的位置。

通过 KMP 算法，可以迅速定位词语所在位置。获得词语位置之后，只需通过 HTML 的 CSS 控制文字显示颜色即可。



9.4.3 网页快照

网页快照是搜索引擎必备的功能之一，爬虫在获取网页数据之后，会进行网页备份，存放在搜索引擎的文件服务器中。因为收录时间和当前时间不一致，因此，网页快照的内容不代表当前网页的内容，但是网站快照却在某些情况下给予用户不少方便：

(1) 在网页打开较慢甚至网页已经不存在的情况下，可以提供给用户相应参考信息。

(2) 网页快照打开后，网页的内容会根据关键词高亮，高亮显示可以使得用户更加准确地定位信息的所在位置，提高信息查找效率。

值得说明的是，网页快照使得对一些已经“消失”的网页能够进行再现，可能涉及侵权问题。因此，爬虫在进行数据备份之前，需要识别网页是否指定禁止快照，在 HTML 的“meta”中有标识，如下语句所示：

```
<meta name="robots" content="noarchive">
```

上述语句表示告诉爬虫，可以索引此网页，但是请不要在搜索结果中显示网页快照。禁止索引的语句如下：

```
<meta name="robots" content="noindex">
```

9.5 搜索智能提示

搜索引擎搜索框的智能提示是任何一个搜索的标配，目的有两方面：一方面是防止用户输错关键词，纠正用户搜索词；另一方面是引导用户到最佳搜索条目上。但智能搜索提示的最终目的是提供优质的搜索服务体验。利用搜索框进行智能提示，至少要满足允许用户使用汉字和拼音输入、支持即时响应提示、

允许在多音字下进行纠错并提示、支持拼音简写提示、历史记录优先出现在提示框中等需求。

(1) 允许用户使用汉字和拼音输入。使用汉字输入理所当然，但是使用拼音输入也必须支持，例如“**changanjie**”能够准确提示“长安街”相关提示。

(2) 支持即时响应提示。如果等待用户输入完成之后，才能给予提示，有可能达不到提示效果。例如，用户意图搜索“紫禁城”时，在输入前缀“紫禁”时，能够产生提示“紫禁城”。

(3) 允许在多音字下进行纠错并提示。汉字在不同的词语中，存在读音不一致的情况。例如，搜索“**ruchouweigan**”时，能够正确提示“乳臭未干”，其中的臭为多音字(xiù、chòu)，可参考9.4节实现。

(4) 支持拼音简写提示。例如，用户搜索“**hzw**”，希望能够提示“海贼王”。

(5) 历史记录优先出现在提示框中。比如用户搜索过“购买机票”，那么下次当用户输入完“购买”二字之后，提示中优先显示“购买机票”，如果还搜索过“购买火车票”，则根据搜索频率，确定优先显示的提示。

智能提示搜索词并不是根据一成不变的词典而来的，提示词实质上是基于所有用户的搜索历史记录产生的，用户在互联网上的搜索都具有相似性。对庞大的用户群提交的搜索词，进行热度排行，从高到低就产生了相应的提示词。这些提示词被按照一定的字典树的数据结构存储起来，进行智能提示。例如，对于搜索热词“三国志”“三国风云”“三峡游记”“三峡游览”，在相同时间范围内分析搜索热词的搜索词频，并将其转换为汉语拼音及拼音简写，如表9-13所示。

表 9-13 搜索热词与搜索词频、汉语拼音、拼音简写对照表

搜索热词	搜索词频	汉语拼音	拼音简写
三国志	100	san guo zhi	sgz
三国风云	100	san guo feng yun	sgfy
三峡游记	120	san xia you ji	sxyj
三峡游览	110	san xia you lan	sxyl

通过表 9-13，对搜索热词的汉语构建字典树，如图 9-26 所示。

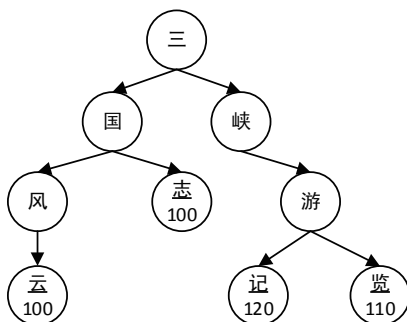


图 9-26 基于搜索热词构建的字典树

基于图 9-26 所示的字典树，当用户仅输入“三”字时，会将字典树节点“三”的所有词语的节点获取出，并根据搜索词频进行排序输出，分别为“三峡游记”“三峡游览”“三国志”“三国风云”。但是在遇到热词的搜索词频相同的情况下，例如用户在输入“三国”时，首先根据字典树找到节点“三”，然后在节点“三”的子节点中找到节点“国”，并将节点“国”的所有叶子节点输出分别为“三国风云”“三国志”，然后两者的搜索词频一致，则按照字符串由短开始输出，所以优先输出“三国志”，其次才是“三国风云”。

对于汉语拼音的输入和拼音简写的输入，也需要构建对应的汉语拼音字典树及拼音简写字典树，如图 9-27 所示。

由于汉语拼音和拼音的简写表达能力相比热词的汉字字典树过于模糊，所以它们的存储结构会有一些改变。在热词的汉字字典树中，每个节点只需标识

当前节点是否是一个词即可，而对于汉语拼音字典树与拼音简写字典树，需要存储这个节点可能表达的含义，以列表的形式存在，并按照搜索词频由高到低排序。例如，拼音简写“sxyj”可能表达的是词语“三峡游记”或者“三险一金”，当用户输入拼音简写“sxy1”时，则定位到最后的节点“1”之后，将节点“1”对应的含义列表输出即可，按照搜索词频依次为“三峡游览”“三星医疗”“三鑫医疗”。

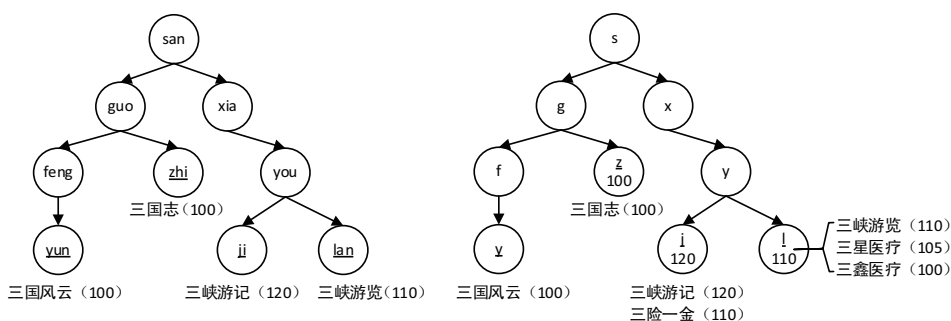


图 9-27 热词的汉语拼音字典树（左）及拼音简写字典树（右）

值得进一步说明的是，对表 9-13 中的汉语拼音，用户的输入不一定也是按照空格分隔好的。例如，“三国风云”，正常的输入处理拼音是“san guo feng yun”，但用户的输入可能是“sanguofengyun”。在处理此类连续拼音时，需要结合基于词典分词的方法（3.2 节有介绍），将所有的汉语拼音作为词库，然后基于词库对输入进行切分即可。也可以按照“sanguofengyun”的每个英文字母建立字典树，然后直接进行字典树匹配，这样做虽然避免了拼音切分的问题，但是会导致字典树层级较大，对性能有一定影响，在工程应用中，根据实际情况两种方案选择一种即可。

由于需要具备实时提示功能，因此同时设计前端页面和后台服务器的请求，需要通过前端异步请求数据，并将异步返回的数据显示到页面搜索提示框中。

Ajax（Asynchronous JavaScript and XML）是智能提示常用的呈现技术，是



通过异步的方式，向后台发送智能提示请求，而前端依然保持不变。传统的 Web 应用会通过表单的方式向服务器发送请求，此种方式不仅浪费宽带资源也不符合用户体验，会有较长的时间延迟。通过 Ajax 技术可以在页面保持用户访问的情况下，在后台获取新的数据，并动态呈现。

智能提示作为用户使用的核心功能之一，也必须能够及时响应互联网的热词提示，例如，用户输入拼音“yunbeitai”，能够及时提示出“云备胎”，需要根据实际情况进行热词统计更新。

9.6 网页排序

网页排序是整个搜索服务中最重要的组成部分。有效的排序算法是搜索质量重要的评价指标之一。网页排序是基于综合性因素的结果，包括搜索相关性、网页信任值、链接评价等。

9.6.1 基于 PageRank 的网页重要性评价

PageRank 是著名的网页链接评价算法，用以体现网页链接的重要性。其思想是通过网页之间的投票得出网页在互联网中的重要性，投票方式则是链接的链向关系。倘若网页 P_1 指向网页 P_2 ，则可以认为， P_1 认为 P_2 相对比较重要。从而把 P_1 的一部分重要性赋予 P_2 ，也可以认为是 P_1 对 P_2 进行了一次投票，这部分投票值通过数值表示即为 $\text{PageRank}(P_1)/\text{Out}(P_1)$ ， $\text{PageRank}(P_1)$ 表示 P_1 的 PageRank 值，也就是本身的重要性； $\text{Out}(P_1)$ 表示 P_1 的外链接总数。而 P_2 的 PageRank 值为所有指向它的网页投票值之和。

举例说明，假设某个局域网只有 4 个网页 A 、 B 、 C 、 D 。如果 B 、 C 、 D 页面都存在链接指向 A ，那么， A 的 PageRank 值为 B 、 C 、 D 的 PageRank 之和，

即 $\text{PageRank}(A) = \text{PageRank}(B) + \text{PageRank}(C) + \text{PageRank}(D)$ ，如图 9-28 所示。

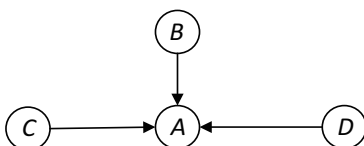


图 9-28 单一关系下网页重要性计算

如果在图 9-28 的基础上，使得 D 页面同时指向 B 、 C 页面， B 页面也指向 C 页面，关系如图 9-29 所示。

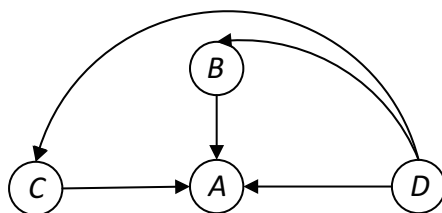


图 9-29 复合关系下网页重要性计算

根据图 9-29 所示，则 $\text{PageRank}(A) = \text{PageRank}(C) + \text{PageRank}(B)/2 + \text{PageRank}(D)/3$ 。

根据图 9-28 和图 9-29 进行归纳，PageRank 的公式将会转换为如下形式：

$$\text{PageRank}(A) = \left(\frac{\text{PageRank}(B)}{\text{Out}(B)} + \frac{\text{PageRank}(C)}{\text{Out}(C)} + \frac{\text{PageRank}(D)}{\text{Out}(D)} + \dots \right) d + \frac{1-d}{N}$$

标准公式化为：

$$S(V_i) = (1-d) + d^* \sum_{j \in \text{In}(V_i)} \frac{1}{|\text{Out}(V_j)|} S(V_j)$$

其中， $S(V_i)$ 表示词在文章中的重要性，类似 PageRank 值。 d 为阻尼系数，一般设定阻尼系数值为 0.85。 $\text{In}(V_i)$ 表示指向网页 i 的链接网页集合。 $|\text{Out}(V_j)|$ 是网页 j 中的链向外部的链接个数。

Google 在提出这个公式的时候，设定的每个页面最小值为 $1-d$ 。 d 是阻尼系数。因此每个页面的 PageRank 是基于链向其的页面 PageRank 计算得出的，通过不断迭代最终达到收敛，以确定每个页面的 PageRank。

针对标准化的 PageRank 公式，计算图 9-30 中的所有网页的重要性，默认初始状态 A 、 B 、 C 、 D 、 E 网页的重要性均相等，且为 1.0。

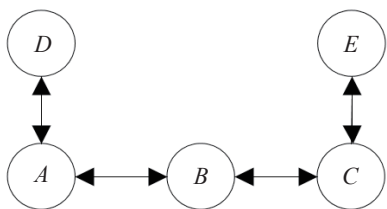


图 9-30 标准 PageRank 公式下计算网页的重要性

首先进行第一轮迭代运算：

$$\begin{aligned}\text{PageRank}(A) &= (1-0.85) + 0.85 * (\text{PageRank}(D)/1 + \text{PageRank}(B)/2) \\ &= 0.15 + 0.85 * (1/1 + 1/2) \\ &= 1.425\end{aligned}$$

$$\begin{aligned}\text{PageRank}(B) &= (1-0.85) + 0.85 * (\text{PageRank}(A)/2 + \text{PageRank}(C)/2) \\ &= 0.15 + 0.85 * (1.425/2 + 1/2) \\ &= 1.180\ 625\end{aligned}$$

$$\begin{aligned}\text{PageRank}(C) &= (1-0.85) + 0.85 * (\text{PageRank}(B)/2 + \text{PageRank}(E)/1) \\ &= 0.15 + 0.85 * (1.180\ 625/2 + 1/1) \\ &= 1.501\ 765\ 625\end{aligned}$$

$$\begin{aligned}\text{PageRank}(D) &= (1-0.85) + 0.85 * (\text{PageRank}(A)/2) \\ &= 0.15 + 0.85 * (1.425/2) \\ &= 0.755\ 625\end{aligned}$$

$$\text{PageRank}(E)=(1-0.85)+0.85*(\text{PageRank}(C)/2)$$

$$=0.15+0.85*(1.501\ 765\ 625/2)$$

$$=0.788\ 250\ 390\ 625$$

依次不间断迭代上述过程，直到值收敛，最终结果如表 9-14 所示。

表 9-14 基于标准 PageRank 公式的网页重要性计算结果

网页	初始值	第 1 次迭代	第 1000 迭代
A	1.0	1.425	1.229 729 729 729 73
B	1.0	1.180 625	1.195 270 270 270 27
C	1.0	1.501 765 625	1.229 729 729 729 73
D	1.0	0.755 625	0.672 635 135 135 135
E	1.0	0.788 250 390 625	0.672 635 135 135 135

由于图 9-30 是一个对称结构，因此从直观角度来看，网页 D 和网页 E 的重要性应该相等且相对较低，而网页 A 和网页 C 的重要性也应该相等且相对较高。由表 9-14 的计算结果可知，PageRank 标准化公式验证了直观感觉，即使是第一次迭代计算之后并不完全符合预期，但是通过不断迭代结果不断符合预期，直至收敛。

值得注意的是，并不是每个链接页面都需要加入进行 PageRank 计算，网页 HTML 代码中若设置如下信息：

```
<meta name="robots" content="nofollow" />
```

表示告诉爬虫，请不要跟踪当前网页的外部链接，通俗来讲即请不要访问本网页的出站链接，不必纳入 PageRank 计算体系。

9.6.2 基于 Hits 算法的网页权威性评价

网页权威性考察的是网页是否具备高质量，以及网页在互联网中的口碑情况。网页越权威则信息可靠度越高。Hits (Hyperlink-Indexed Topoc Search) 算法是一



个比较成熟可用的网页权威性评价算法。它是基于链接的分析算法，与用户搜索的主题息息相关，算法核心集中在权威值（Authority）和枢纽值（Hub）两方面。

（1）权威值。按照一般思维，若一个网页是多个其他网页的外链，则该网页理论上讲比较重要，但一个网页不是其他网页的外链，只被一个非常重要的网页作为外链，则该网页也有可能是非常重要的网页，也就是权威网页传递的连接，也有可能是比较权威的网页。

（2）枢纽值。某个网页指向了很多权威性网页，但该网页可能并不是权威网页，甚至指向它的网页也并不多，因为任何一个网站开发者均可制作这样的网页。

对于一个网页来说，其权威值是所有指向该网页的枢纽值之和，其枢纽值是该页面外链页面的权威值之和，基于此，一个具有较高权威值的页面很可能是多个高枢纽值的页面的外链，一个具备较高枢纽值的页面，外链中会包含很多权威值较高的页面。

因此，Hits 算法利用主题来衡量网页的权威程度。针对不同主题，相同的网页重要程度也是不相同的。研究还发现，同一主题下的高权威值并不存在相互的连接关系，几乎都是通过枢纽值高的网页链接。例如主题为“新闻”类的网站，它们之间很少相关链接，几乎没有看到过新浪新闻链接搜狐新闻，搜狐新闻链接网易新闻，但是在导航网站中，它们都同时出现。此种情况可以认为导航网站即是枢纽值较高的网站，各大新闻网站即是新闻主题下具备较高权威值的网站。

Hits 算法流程包括如下三个方面。

（1）建立根集合。根集合是用户搜索后产生的与主题搜索相关的页面集合。

（2）建立扩展集合。扩展集合是搜索结果页面的所有指向它和它的外部链接集合。

(3) 迭代计算权威值, 枢纽值。根集合和扩展集合组成一个图结构, 基于一个网页其权威值是所有指向该网页的枢纽值之和, 其枢纽值是该页面外部链接页面的权威值之和, 在初始状态中, 每个页面的枢纽值和权威值都仅为 1, 进行不断的迭代计算。计算出与主题相关的最佳权威页面。

值得注意的是, Hist 算法和 PageRank 比较类似, 它们都充分利用链接之间的相互关系, 挖掘有效信息, 但是二者侧重点不同。

(1) Hist 算法的计算是以主题为中心, 与用户的搜索词密不可分。PageRank 具备的是垃圾信息淘汰机制, Hist 具备的是主题不相关淘汰机制。

(2) Hist 算法既然是在用户产生搜索行为之后的计算, 则必须是实时计算, 而 PageRank 在后台运行, 属于离线计算。

Hist 算法相对计算量小于 PageRank, PageRank 需要尽可能覆盖到更多的网页, 但也是因为基于此, PageRank 算法的稳定性较高, 一个网站的评分基本上不会在较短时间内发生重大变化, 而 Hist 算法则存在这样的情况, 一个链接的变化对通过 Hist 算法计算后的权威值排名都会有很大影响。

9.6.3 Hilltop 算法

Hilltop 算法实质上是对 PageRank 及 Hist 算法的综合。依然沿用 PageRank 的链接分析法, 同时也将 Hist 算法中的主题为中心的思想融合, 类似于若是两个或两个以上主题相关性较强的网站同时链接到另外一个网站, 则认为该网站在该主题性下可靠性更高。这样做可以有效减少网站之间随意无关链接导致的链接权重误差。

为保证将主题相关性纳入排序中, Hilltop 算法会对特定的部分网页文档进行整理归纳, 这部分网页相对整体质量较高、特定主题针对性强, 其次文档相

互之间属于非附属网页，此类文档称为“专家文档”。需要约束专家文档属于非附属网页的原因在于，专家页面链接权重分析相对关系较重，容易产生裙带关系。在用户搜索过程中，Hilltop 算法的流程如图 9-31 所示，首先通过用户的搜索词筛选出专家文档，并对专家文档进行评分；其次将搜索词与网页搜索的倒排索引匹配，获得搜索结果页面；然后利用获得的专家文档与搜索结果页面的链接关系，分析链接权重；最终获得搜索结果页面的链接评分。

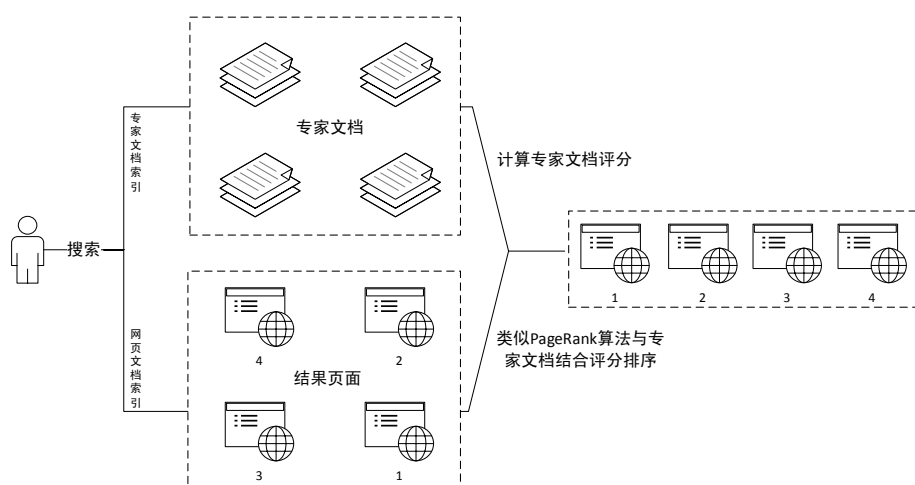


图 9-31 Hilltop 算法的流程

在上述过程中，专家文档对于整个搜索和排序起到非常重要的作用。Hilltop 算法同 Hist 算法一样，需要通过在线实时计算获得。

9.6.4 网页作弊评价

搜索引擎一直致力于数据纯净化，给用户最佳的搜索体验。但是实际中，不少网站会仔细分析搜索引擎排序，试图通过非正当手段，使得自己的网站排名尽可能靠前，即使与搜索词不相关。随着互联网的发展，以商业化运作的网站远远超过了知识共享的免费网站，因此，这些商业化网站视图通过作弊的方式希望尽可能将搜索流量导入到该网站中，搜索引擎允许网站通过优化，使得

命中最佳效果，但是通过作弊手段诱导用户，是搜索引擎所不允许的。网站站长通过基于链接的网页作弊方式、基于内容的网页作弊方式、基于隐藏信息的网页作弊方式进行作弊。

(1) 基于链接的网页作弊方式。之所以会对链接作弊，是因为链接是网页本身价值的一个体现，通过有目的地构建网页之间的相互关系，从而提升页面在搜索引擎中的重要性评分。

(2) 基于内容的网页作弊方式。内容是搜索相关性参考的重要指标之一，作弊者通过添加一些无关正文的内容放置到网页中，并且伪装成热点词汇的方式。

(3) 基于隐藏信息的网页作弊方式。隐藏信息的方式是欺骗搜索引擎和浏览器的方式，通过 CSS 样式控制页面内容是否显示，因此作弊者可以为所欲为地新增很多内容，但是这部分内容对于该页面没有相关性，更有作弊者甚至通过页面跳转的方式进行作弊。

作弊者试图通过各种手段达到站点排名靠前的原因在于背后的经济利益链，主要是如下几方面：

(1) 通过页面的广告获得经济利益。

(2) 用户一旦点击此类页面，很可能会被安装一些恶意插件、软件，甚至对用户的个人信息造成安全威胁。

(3) 实质网站为一些钓鱼网站、诈骗网站等，会对用户的财产安全形成威胁。

它们的运作模式主要依靠内容农场 (Content Farm)。为获得更高的网络流量，使得利益最大化，站长和一些网络广告公司，利用各种手段改造网页内容，例如，大量复制质量较高的网络热文，通过修改其中的部分内容，使得搜索流量流向自己的网页，还会利用当下搜索热词，在质量较差的网页中插入相关热词，也试图使搜索排名尽可能靠前，获得用户的点击，谋求网络广告的收益。



通过内容农场创作出来的文章，数据质量参差不齐，参考价值也不高，掺杂着大量广告内容，网页布局也不符合用户体验。

针对内容农场式的网页作弊，可以通过基于信任模型、不信任模型等各类算法模型进行分析。

(1) 基于信任模型的 TrustRank。TrustRank 是斯坦福大学和 Yahoo 的一项联合研究，利用这种方法对网页进行信任值评分，事先确定一些信任值较高的站点作为种子网页，通过种子网页的外链，将信任值不断向外播种，直到最后对所有的网页进行信任值汇总，通过信任值的方式排除掉信任值较低的站点。TrustRank 虽然理论上设计比较合理，但在工程中却带来了新的问题。一方面对于刚刚新建的站点，即使内容保持更新频率较高，而且均为原创，也很难拿到较高的排名；另一方面搜索结果中大部分都是权威站点的结果，但其内容不完全是原创。

(2) 不信任传播模型。和 TrustRank 相反，先找到一堆作弊的网页，然后通过这些网页的外链分析其不信任评分，不同点在于 TrustRank 的信任分值是正向传递，而不信任传播模型是反向传递。例如，网页 P_1 已经被确定为不信任网页，则 P_1 外链中的网页 P_2 比链接向 P_1 的网页 P_3 信任值高，这就是反向传递。

(3) Panda 算法。Google 通过 Panda 算法解决内容农场导致的问题，Panda 算法的核心思想在于通过页面内容、链接概况及网页在搜索结果中的点击情况综合考虑分析。百度通过用户对网页的评价方式惩罚包括采用内容农场方式运营的各类垃圾网站。

(4) 网页特征判定模型。通过分析作弊网页的特征，例如，通过网页的链入链接和外部链接的数量分布情况及网页外链中的死链与活链的比例，死链是指服务器的地址已经改变或者网页地址已经无法再访问，活链则表示可以正常访问的链接；还可以使用作弊网页的 url 长度分布、语句特征等。

9.6.5 网页排序调试

在搜索引擎研发过程中，常常会有新的算法迭代，尤其是排序算法。有时，研发人员也试图探索一些自动化测试技术，验证新算法的优劣，而常用的就是 AB 测试，俗称“灰度”。

对于同一个功能算法，开发人员会准备两个甚至更多的方案，每个方案都安排不同用户使用，记录下用户的使用情况，用以确定哪个方案最佳。例如，搜索引擎结果排序算法上，准备一套新的算法方案 *B*，在技术调研阶段确定目前已经上线的方案 *A* 在理论上优于方案 *B*，但是方案 *B* 在另外一个项目中的表现优于方案 *A*，开发人员尝试将方案 *B* 借鉴过来，那么如何确定搜索引擎中方案 *A* 与方案 *B* 的优劣？目前已知，在方案 *A* 中相关性结果靠前，用户个性化的结果靠后，而方案 *B* 恰恰相反，用户个性化的结果靠前，相关性结果靠后。从目前已经拥有的用户中取 5% 使用方案 *B* 的搜索结果，这些用户都是随机产生的，也不知道会有这样的差异，让用户使用一周时间，记录用户点击网页的情况，根据相同搜索词，判断链接在方案 *A* 中及方案 *B* 中的点击分布情况。点击分布越靠前，说明方案越优。在搜索引擎早期，排序调试往往由核心开发人员来感知排序，但这样会导致结果缺乏客观性依据。采用众包的思想，利用用户来帮助工程师调试搜索引擎算法，是搜索引擎在成熟阶段的必用手段，必应搜索曾经利用 AB 测试，让用户感受是必应的搜索效果好，还是 Google 的搜索效果好。

众包的思想非常不错，值得借鉴的案例还有浏览器上的广告拦截插件，对于广告拦截插件它本身并不知道页面中的某个元素是广告，但是如果众多网民都通过手动方式把某个元素屏蔽，则基本上可以确认，此元素是广告，系统再推送给其他用户，其他用户无须手动清理广告，自动屏蔽，这也是为什么有的产品，用户越多，用的效果也越好。

通过大众参与的网页排序调试方法，可以使得搜索引擎排序质量得到进一步提升。

9.7 个性化搜索

个性化搜索是现代搜索引擎的标配。在搜索引擎的个性化搜索过程中，需要充分保证搜索的质量和效果，将利用基于相关性和协同过滤推荐结合起来可以充分发挥其各自的优势为系统提供更高的准确率。

9.7.1 个性化搜索示例

由于二者都将分别获取到网页信息，然而按照其各自获取网页信息的规则，最终搜索结果网页集合合并的时候，排序规则则需要重新考虑。经过不断调整和优化，给予相关性权重 0.8，个性化推荐网页权重 0.2，相同网页合并权值。通过权值分配，搜索结果中相关性搜索的结果优先显示，个性化推荐网页滞后显示，两者重合的页面最先显示，如表 9-15 所示。

表 9-15 网页个性化搜索结合示例

网页编号	相关性评分	个性化评分	综合评分
P_1	0.8	—	$0.8*0.8=0.64$
P_2	0.7	0.7	$0.7*0.8+0.7*0.2=0.7$
P_3	0.6	0.3	$0.6*0.8+0.3*0.2=0.48$
P_4	—	0.4	$0.4*0.2=0.08$
P_5	0.3	—	$0.3*0.8=0.24$

基于表 9-15，最终网页排序为 P_2 、 P_1 、 P_3 、 P_5 、 P_4 ，其中， P_2 和 P_1 在相关性评分中的位置和综合评分中的位置不同即是对个性化的体现。

9.7.2 人工神经网络与个性化搜索

搜索结果的个性化围绕用户特征进行分析。通过用户的搜索历史记录和点击行为，以及相似用户分析可以获得相应用户特征。但是从工程角度来讲，仅仅是通过计算出用户特征，这些特征需要组合成数学模型，通过数学模型进行实际的个性化搜索，采用BP神经网络是一种较好的解决方案。BP神经网络是一种能够进行反向传递误差，却能够修正误差的多层映射网络，是目前应用最广泛的神经网络模型之一。用于搜索引擎个性化之前，需要准备两大模型，用户特征模型和词语特征模型。

(1) 用户特征模型。通过用户注册信息和用户搜索与点击日志建立，其中包括用户年龄、性别、学历、爱好，通过将特征模型数值化，归一化到0~1之间的值。针对每个用户选取 N 个特征，设定 N 个特征的值在区间 $[0,1]$ 上。例如，性别越接近1表示越可能是男性，越接近0表示越可能是女性；学历越接近1，表示学历程度越高；某爱好越接近1，表示此爱好越强烈。

(2) 词语特征模型。是在历史搜索日志中不断积累和分析出的结果。它表示每个词语与用户模型的相关程度，与用户模型存在相同的 N 个特征。

词语特征模型与用户特征模型示例如表9-16所示。

表 9-16 词语特征模型与用户特征模型示例

	性 别	学 历	与计算机类相关度	...	与财经类相关度
某用户“张琪”	0.31（偏向女性）	0.75	0.31	...	0.82
某用户“李非”	0.86（偏向男性）	0.89（偏向高学历）	0.84	...	0.29
词“人工智能”	0.79（男性相关较高）	0.91（偏向高学历）	0.90	...	0.26
词“东方财经”	0.51（与性别关系不大）	0.78	0.41	...	0.96

用户特征模型和词模型实质上是通过训练得出的两个矩阵。值得说明的是，用户模型和词语特征模型并不是一直不变，会进行周期性的重新计算，矩阵对

特征抽取的精度越大，结果越精准，但是特征选取得越多，计算就越复杂。

BP 神经网络的作用是结合用户本身及其搜索词语给予最佳相关的网页筛选和重排序，例如，用户“李非”搜索“人工智能”更可能是了解人工智能专业知识，因此人工智能专业的网页排序较高，但是对于用户“张琪”搜索“人工智能”，多项指标表明她更多的是初步了解人工智能领域，在相关性一致的情况下，对于介绍性的文档应该优先排序，专业性的文档靠后排序。BP 神经网络模型如图 9-32 所示。

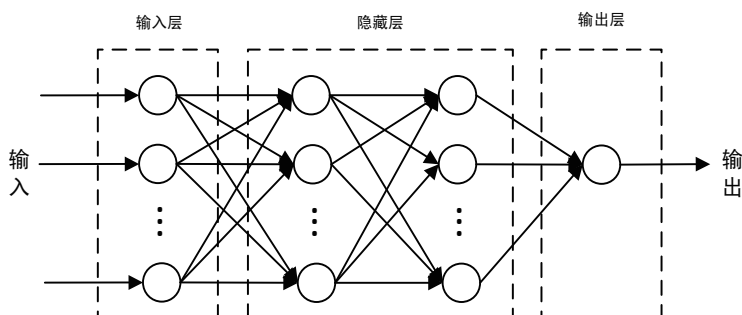


图 9-32 BP 神经网络模型

对于相关性给出的每一个网页，取其网页的关键词，每个关键词对应的词语特征与用户特征相乘作为输入，输出为网页的个性化权重。将个性化权重与相关性、PageRank、HillTop 算法的综合权重进行加权求和，得出最终既满足相关性也满足权威性和个性化的结果。

9.7.3 地理位置搜索

基于地理位置的附近地点搜索，是当前搜索引擎中非常重要的组成部分。随着目前拥有 GPS 的智能设备越来越普及，使得基于地理位置的搜索也日益增多。例如，用户搜索“最近必胜客餐厅”，则用户期望获得离自己最近的必胜客餐厅。

用户可以通过 IP 地址、HTML 5、移动客户端等方式获得用户的地理位置。

(1) IP 地址。搜索引擎可以通过 IP 地址大致获得用户所在位置。

(2) 通过 HTML 5。在 HTML 5 中，允许通过 JavaScript 直接获得用户当前所在的经纬度，如下代码所示。

```
<script>
var x = document.getElementById("location");
function getLocation() {
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(showPosition);
    } else {
        x.innerHTML = "Geolocation is not supported by this browser.";
    }
}
function showPosition(position) {
    x.innerHTML = "Latitude: " + position.coords.latitude + "<br/>Longitude: " + position.coords.longitude;
}
</script>
```

被获得的页面中会提示“这些网站正在跟踪您的位置”，如图 9-33 所示。



图 9-33 基于 HTML 的地理位置获取提示

(3) 通过移动客户端位置获取。任何 Android 或者 iPhone 等智能手机，基本上都有地理位置服务功能。

以用户搜索“最近必胜客餐厅”为例，当获得用户地理位置之后，通过搜索引擎也获得若干必胜客餐厅的地理位置，按照正常的思路，只需通过经纬度计算用户与餐厅的距离即可。要求从球面的一个点 $A(x_a, y_a)$ 出发到球面的另

一个点 $B(x_b, y_b)$ 的最短路径值，可以利用 Haversine 公式，如下公式所示，其中 r 是地球半径。

$$d=2r*\arcsin\left(\sqrt{\sin^2\left(\frac{x_b-x_a}{2}\right)+\cos(x_a)\cos(x_b)\sin^2\left(\frac{y_b-y_a}{2}\right)}\right)$$

将所有的必胜客餐厅与用户的地理位置通过 Haversine 公式计算距离，理论上可行，但是实际中，由于计算量过大，会导致性能问题。

在工程应用中，采用 Geohash 算法，对地理位置的经纬度进行编码处理。Geohash 算法的核心思想是将二维的经纬度编码成一维的字符串信息。例如，对于某位置，纬度和经度分别为 29.546 066 2 和 106.524 235 499 999 99。首先将纬度区间 $(-90,90)$ 分成两个区间 $(-90,0)$ 和 $[0,90)$ ，如果给定的目标位置的纬度位于右部区间 $[0,90)$ ，则编码为 1，否则编码为 0。然后再将 $[0,90)$ 划分为 $[0,45)$ 和 $[45,90)$ ，再次判断目标位置的纬度位于哪个区间，左部区间编码为 0，右部区间编码为 1。迭代上述过程，直到约定的足够精确的位置，本例则采用 20 位编码精度，如表 9-17 所示。

表 9-17 采用 Geohash 算法的纬度编码过程

经度区间	左部区间	右部区间	编码值
$(-90,90)$	$(-90,0)$	$[0,90)$	1
$[0,90)$	$[0,45)$	$[45,90)$	0
$[0,45)$	$[0,22.5)$	$[22.5,45)$	1
$[22.5,45)$	$[22.5,33.75)$	$[33.75,45)$	0
$[22.5,33.75)$	$[22.5,28.125)$	$[28.125,33.75)$	1
$[28.125,33.75)$	$[28.125,30.9375)$	$[30.9375,33.75)$	0
$[28.125,30.9375)$	$[28.125,29.53125)$	$[29.53125,30.9375)$	1
$[29.53125,30.9375)$	$[29.53125,30.234375)$	$[30.234375,30.9375)$	0
$[29.53125,30.234375)$	$[29.53125,29.8828125)$	$[29.8828125,30.234375)$	0
$[29.53125,29.8828125)$	$[29.53125,29.70703125)$	$[29.70703125,29.8828125)$	0
$[29.53125,29.70703125)$	$[29.53125,29.619140625)$	$[29.619140625,29.70703125)$	0

续表

经度区间	左部区间	右部区间	编码值
[29.531 25,29.619 140 625)	[29.531 25,29.575 195 312 5)	[29.575 195 312 5,29.619 140 625)	0
[29.531 25,29.575 195 312 5)	[29.531 25,29.553 222 656 25)	[29.553 222 656 25,29.575 195 312 5)	0
[29.531 25,29.553 222 656 25)	[29.531 25,29.542 236 328 125)	[29.542 236 328 125,29.553 222 656 25)	1
[29.542 236 328 125,29.553 222 656 25)	[29.542 236 328 125,29.547 729 492 187 5)	[29.547 729 492 187 5,29.553 222 656 25)	0
...	

因此实现对纬度 29.546 066 2 的编码，结果为 10101010000001010110。按照同样的方法，对经度也进行编码，经度的区间为 (-180,180)，编码过程如表 9-18 所示。

表 9-18 采用 Geohash 算法的经度编码过程

纬度区间	左部区间	右部区间	编码值
(-180,180)	(-180,0)	[0,180)	1
[0,180)	[0,90)	[90,180)	1
[90,180)	[90,135)	[135,180)	0
[90,135)	[90,112.5)	[112.5,135)	0
[90,112.5)	[90,101.25)	[101.25,112.5)	1
[101.25,112.5)	[101.25,106.875)	[106.875,112.5)	0
[101.25,106.875)	[101.25,104.062 5)	[104.062 5,106.875)	1
[104.062 5,106.87 5)	[104.062 5,105.468 75)	[105.468 75,106.875)	1
[105.468 75,106.875)	[105.468 75,106.171 875)	[106.171 875,106.875)	1
[106.171 875,106.875)	[106.171 875,106.523 437 5)	[106.523 437 5,106.875)	1
[106.523 437 5,106.875)	[106.523 4375,106.699 218 75)	[106.699 218 75,106.875)	0
[106.523 437 5,106.699 218 75)	[106.523 437 5,106.611 328 125)	[106.611 328 125,106.699 218 75)	0
[106.523 437 5,106.611 328 125)	[106.523 437 5,106.567 382 812 5)	[106.567 382 812 5,106.611 328 125)	0
[106.523 437 5,106.567 382 812 5)	[106.523 437 5,106.545 410 156 25)	[106.545 410 156 25,106.567 382 812 5)	0



续表

纬度区间	左部区间	右部区间	编码值
[106.523 437 5,106.545 410 156 25)	[106.523 437 5,106.534 423 8 28 125)	[106.534 423 828 125,106.54 5 410 156 25)	0
[106.523 437 5,106.534 423 8 28 125)	[106.523 437 5,106.528 930 66 4 063)	[106.528 930 664 063,106.534 423 828 125)	0
...	

通过表 9-18 计算得出经度的编码结果：11001011110000000010。

在实现对经纬度进行编码之后，将经纬度进行合并。合并规则按照奇数位存放纬度，偶数位存放经度，得到编码 1110010011001110101000000001000100 011100。

最后对二进制编码进行字符串编码，以 0~9 十个数字及 22 个英文字母（移除字母 a、i、l、o）对二进制编码进行 base32 编码，编码对应关系如表 9-19 所示。

表 9-19 十进制与 base32 编码对照表

十进制	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Base32	0	1	2	3	4	5	6	7	8	9	b	c	d	e	f	G
十进制	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Base32	h	j	k	m	n	p	q	r	s	t	u	v	w	x	y	z

将二进制编码按照从前到后，五位一组进行 base32 编码，如图 9-34 所示。

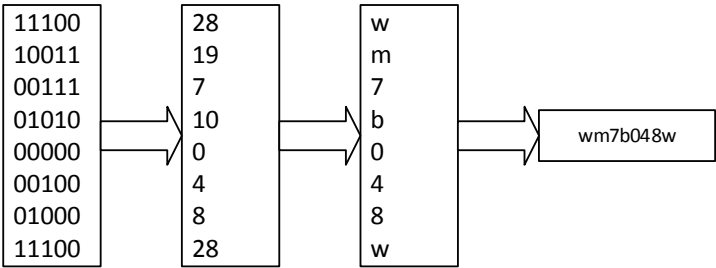


图 9-34 对经纬度进行 base32 编码

因此最终获得坐标（29.546 066 2，106.524 235 499 999 99）的编码值 wm7b048w，随着编码精度越高，则编码之后的字符串越长。这样的编码值对

于地理位置搜索的意义在于，越是相近的区域，它们的坐标进行编码之后，编码的前缀越相似。Geohash 实质上是空间索引技术的一种方式，适用于坐标点这样的数据。

获得用户的地理位置仅仅是实现搜索的第一步，在搜索结果中，还需要结合距离和相关性排序。

9.8 图片搜索

图片搜索是目前搜索引擎中搜索流量仅次于网页搜索的多媒体搜索，它主要包括基于内容的图片搜索及基于文本的图片搜索。

9.8.1 基于内容的图片搜索

基于内容的图片搜索（Content-Based Image Retrieval）主要思想是基于图片本身拥有的信息进行搜索，在给定查询图片的情况下，进行图片搜索，是“以图搜图”的应用搜索。通过图片搜索获得相似图片，主要采用感知哈希算法实现（Perceptual Hash Algorithm），核心思想是通过对每张图片构建唯一指纹，图片中指纹越相近则说明图片相似度越高。

（1）尺寸缩放。将所有图片数据进行尺寸缩放，缩放到 $8 \times 8, 64$ 个像素大小。尺寸缩放的目的在于避免图片中一些细节及图片大小对图片搜索的干扰。

（2）色彩简化。将被缩放后的图片数据简化其色彩，进行 64 级灰度，使得整个图片中仅包含 64 种颜色。

（3）灰度平均值。计算每个图片中 64 个像素的灰度平均值。

（4）灰度比较。将 64 个像素的灰度值与平均灰度值进行比较，大于等于

平均灰度值的像素设定为 1；小于平均灰度值的像素设定为 0。

(5) 计算哈希指纹。在灰度进行比较完毕之后，64 个像素组成一个 64 位的整数，这个整数被视为当前图片的指纹。

通过上述过程获得图片指纹之后，只需将用户提交的图片按照同样的方式获得哈希指纹之后，进行海明距离计算，从而获得图片与图片之间的相似度。一般情况下，如果海明距离小于等于 5 则说明图谱具有一定的相似度，若海明距离大于 0，则表明两张图片之间存在一定差异。

采用感知哈希算法的过程，比较简单，能够精确地以图片搜索图片。但是对于一些模糊的图谱，或者图片中存在的一些更改情况，则不能很好地识别出相似图谱。在工程应用中，借鉴感知哈希算法，利用图片的颜色分布情况及内容特征进行图片搜索。

9.8.2 基于文本的图片搜索

基于文本的图片搜索，是通过获取图片附近的文本信息，这些文本信息和网页搜索的文本信息一样，被建立倒排索引，然后通过对倒排索引的使用获得对应图片信息。基于文本的图片搜索的实质与网页搜索类似，它们都是对文件建立相关索引，网页搜索对应的是文档集合，图片搜索对应的是图片的集合，如图 9-35 所示。

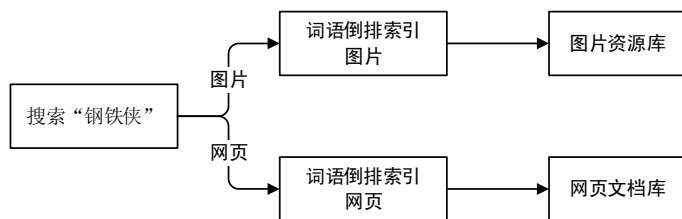


图 9-35 图片搜索与网页搜索中的相同与不同

而对于基于文本的图片搜索，文本信息主要来自三个方面。

(1) 网页 HTML 中的标签，如图 9-36 所示，在 HTML 标签 “img” 中的属性 “alt” 包含的信息，是对该图谱的一种简短描述。

```
<li style="display:none">
  <a href="http://news.ifeng.com/a/20151129/46439313_0.shtml" target="_blank">
    
  </a>
</li>
```

图 9-36 HTML 标签中关于图片的描述信息

(2) 图片周围的信息。图片一般嵌套在网页中某个区域性位置，但是这个区域性位置一般用于讲述该图谱的相关信息，如图 9-37 所示，图中下面一行文字是对该图片信息的一个描述，一般它们位于同一个 HTML 的 “DIV” 标签或者相邻 “DIV” 标签中。



图 9-37 网页中图片附近的文字信息示例

(3) 图片本身的文字信息。为了更加准确地分析图片所描述的信息，学术界一直试图对图片进行光学字符识别（Optical Character Recognition, OCR）。

目前从工程应用的角度来看，主要还是采取前两种方式，对于第三种基于图片上本身文字的识别，理论上是可以获得更加精准的图片描述信息，但是基于如下四个原因，暂时在工程领域没有被广泛使用。



(1) 汉字的 OCR 识别技术准确率还有提升的空间。在 OCR 领域，英文的识别率很高，但是目前中文的识别率较低，容易误识别。

(2) 质量较高的图片中一般不会主动加入大量对图片信息的描述。但是大多数广告图片中存在大量文字，而广告图片一般不作为图片搜索分析的重点。

(3) 图片中存在的自然文字，可能对图片本身所要描述的信息造成干扰。例如，某张图片描述的是某篮球球星，但是他穿的文化衫中包含一些广告文字，而这些广告文字与这位篮球球星却没有直接关系。

(4) 目前基于 HTML 标签及图片周围文字的分析，已经可以达到较好的效果，不必为了追求极限的结果，去付出高额代价。

9.9 搜索与广告

广告是搜索引擎的主要盈利方式。广告投放是搜索引擎根据广告产品特点、服务，确定与其相关的搜索，用户搜索产生广告展示，广告主最终通过广告展示或点击向搜索引擎付费。搜索引擎对广告的投放不同于其他广告媒体，它通过技术手段使得互联网广告业快速发展，具有极强的用户针对性，并且可以跟踪广告效果、几乎面向整个搜索引擎用户群。

基本的广告投放策略，除与网页结果具有相似排序算法之外，还有基于协同过滤的广告推荐方式，其中包括基于 User-Based 协同过滤的广告投放、基于 Item-Based 协同过滤的广告投放及基于 Model-Based 协同过滤的广告投放。

广告投放可帮助用户发现购买需求及产生购买行为。例如，某用户经常搜索“iPhone 6S 最新价”，显然用户是有购买心理的，只是可能因为价钱的原因迟迟未产生购买行为，如果将此信息和电商合作，电商在进行“iPhone 6S”促

销活动时，将会通过搜索引擎广告，将“iPhone 6S”促销信息告知消费者。

9.9.1 广告投放策略

搜索引擎广告成本相对于传统广告价格低廉而且面向的用户群广泛，涉及各行各业的人员，因此，搜索引擎的广告投放策略直接关系到搜索引擎公司的利益。主要从如下几个方面考虑：

(1) 广告关键词确定。广告信息中通过简单的句子表达广告含义，选择广告关键词同搜索网页类似，但是不同点在于需要通过用户对广告的浏览，确定关键词是否有效，甚至需要反馈学习，重新确定最佳关键词。

(2) 搜索与广告相关性算法。搜索网页的排序算法是一个综合性算法，广告展示虽然也是通过搜索产生，但是广告展示的侧重点在于相关性。

(3) 最少开销吸引用户有效点击。广告主通过购买关键词获得广告展示的机会，每一次展示均会付出成本代价，提升广告点击率是搜索广告中需要解决的核心问题。

(4) 精准适配用户。用户的点击行为是不可预知的，为避免用户的无效点击，需要通过不断地了解用户信息，例如性别、兴趣、爱好、历史浏览记录、历史点击记录等相关信息，以提高用户点击率。

9.9.2 基于 User-Based 协同过滤的广告投放

User-Based 是以相似用户的方式进行广告投放的。User-Based 协同过滤方法的基本思想是：如果有三个用户甲、乙、丙，甲用户点击了广告链接 P_1 ，乙用户点击了广告链接 P_1 、 P_2 、 P_3 ，丙用户点击了广告链接 P_1 、 P_3 ，从此认为用户甲、乙、丙三者具有相似的广告点击行为。因为他们三者均点击了广告链接

P_1 。基于此，对于用户甲推荐广告位链接 P_2 、 P_3 ，对于用户丙会推荐 P_2 。例如，针对三个用户对于汽车相关广告的投放，如表 9-20 所示。

表 9-20 基于 User-Based 的广告投放

用户	“奥迪”广告链接	“宝马”广告链接	“奔驰”广告链接
甲	已点击	(可投放推荐)	(可投放推荐)
乙	已点击	已点击	(可投放推荐)
丙	已点击	(可投放推荐)	已点击

对于寻找相似用户的基本思想借鉴了 K 最近邻搜索算法(K-Nearest-Neighbor, KNN)的思想，用 KNN 可以找出某个相互的相似用户群，他们拥有类似的兴趣或广告点击行为。因此，KNN 是根据相似广告商的兴趣进行预测的。

KNN 算法是机器学习中最基本的方法之一，其相似性判断方法采用特征向量空间模型，可以通过计算用户之间的共同特征数量，用于判断该用户是否属于相似用户。过程如下所示：

(1) 确定 K 邻近算法的 K 大小， K 的含义是指搜寻目标数据附近的邻居的个数。

(2) 根据事先确定的距离度量方式（以点击相同广告的数量作为距离度量值）搜寻用户身边距离最近的 K 个用户。

(3) 确定好 K 个邻近用户之后，选取他们点击网页广告的历史记录集合，并按照点击次数从高到低进行排序，移除已经向用户投放的广告，然后依次取相应广告进行展示，如图 9-38 所示。

KNN 是一种类似于投票的方式来决定需要投放的广告，但是这样看似公平民主的算法，有一个缺点：当数据分布相对较为均匀，且密度较为集中时， K 的不同取值，将会导致目标数据的分类波动。要解决这个办法，需要将目标分类数据到各个点的距离作为一种权值参考，例如，具有相似点击广告的用户，向其推荐的广告信息权重越高。

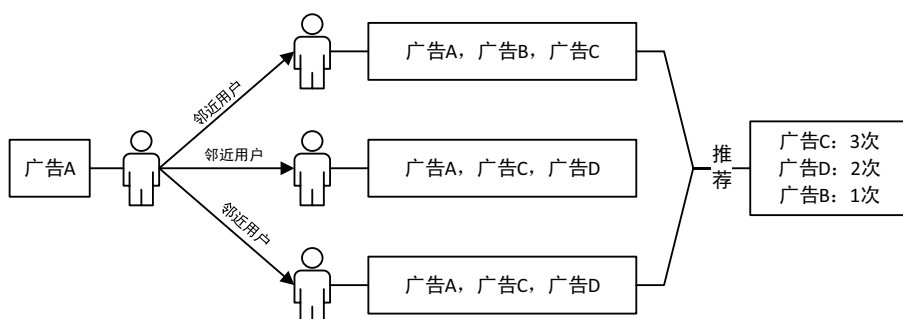


图 9-38 K 邻近用户广告推荐示意

除采用 KNN 的方式进行相似用户获取之外，还可以采用聚类的方式将相似用户聚集在一个信息簇里。处于一个信息簇里面的用户一定是相似的，之所以不采用分类是因为聚类更加灵活，且无须训练。用户数据能够进行聚类，是因为用户不管他们之间有多大的区别，但总有一定的共性，可以通过这些共性对用户聚类，以帮助广告投放。

9.9.3 基于 Item-Based 协调过滤的广告投放

基于 User-Based 的广告投放方式，在用户群逐渐变大之后，随着用户量的不断增加，计算的时间也会逐步变长。因此，试图通过评价机制对广告进行投放，虽然在用户进行广告点击中没有评分系统，但是可以根据用户对相同或相似广告的点击次数来判断用户对此类广告信息的兴趣浓厚程度。基于此，Item-based 广告投放可以根据用户过去对已经点击过的广告评分来构建新的数据库，用于判断用户对其他未展示的广告进行兴趣估值。例如，一个用户频繁点击“手机”相关的广告信息，那么能够依据此信息预测其对电子信息产品具有浓厚兴趣。基于 Item-Based 的广告投放步骤大致如下：

(1) 收集用户点击广告的历史记录。为每个用户建立点击数据库，用于发现用户的兴趣点。



(2) 采用 KNN 计算点击广告与未展示广告的相似度。将用户已经点击过的广告与还未向用户推荐的广告进行相似度计算，对未投放的广告进行评价，得到未投放广告的预测分。多次预测则分值累加，例如，已点击广告 A、B 对未点击广告 C 的预测分值分别是 0.3 和 0.2，则 C 的预测分为 0.5。

(3) 推荐结果展示。按照预测分值从高到底的顺序，向用户推荐未点击的广告信息。

基于 Item-Based 和基于 User-Based 是两种类似的投放方法。一个是基于用户相似程度，另一个是基于历史记录与当前广告相似程度。通过工程实践对比结果得出结论，基于 User-Based 的协同过滤方法整体效果略差于基于 Item-Based 的协同过滤方法。User-Based 更倾向于一种离线分析，而 Item-Based 更类似于一种在线分析。例如，电子商务网站第一次推荐可能为了迅速，使用 Item-Based，第二次，当有离线数据之后，再使用 User-Based 结合 Item-Based 的方法。在通过 Item-Based 投放广告时，不需要采用整个商品集合，只需将分类的产品集合进行操作。例如，将体育类的广告分为一组进行 Item-based 计算。

9.9.4 基于混合模式的广告投放

基于 Item-Based 和 User-Based 协同过滤的投放方式，均是以历史数据为基础的广告投放方式。历史记录在使用过程中会不断积累而变得庞大，导致计算量不断上升。但是历史数据对于广告投放的价值不容否定，因此通过历史记录训练广告投放模型。以 Model-Based 为基础的协同过滤技术包括 Latent Semantic Indexing、Bayesian Networks 等。采用模型的方式，可以实时快速地进行计算。

在工程实践中，广告投放依赖于广告与搜索词相关性、Item-Based、User-Based、Mode-Based 四种混合方式进行广告投放。在投放结果中，分别给予相应权重，如表 9-21 所示。

表 9-21 混合广告投放中的权重设置

投放方式	权值
广告与搜索词相关性	0.65
基于 Item-based 协同过滤	0.15
基于 User-based 协同过滤	0.1
基于 Model-based 协同过滤	0.1

表 9-21 中的权值根据广告类型不同也会相应调整，同时也需要通过广告投放评价进行不断优化调整。在数据量过于少或者用户未产生任何点击的情况下，广告投放结果仅能依靠广告与搜索词相关性分析。

9.9.5 广告投放评价

在投放方式中采用了相关性与协同过滤综合的方式，但是对于搜索引擎广告投放效果的最重要评价指标即广告点击率（Click-Through Rate），在大数据互联网时代搜索引擎除充分挖掘广告本身数据之外，还需要充分挖掘媒体数据，即广告在不同媒体介质（如图片、文字等）、设备终端（如 PC、移动设备等）中产生效果的差异，还有用户数据，以及对互联网网民全面而深刻的认识，通过三者之间的关系将广告投放效果做到最佳。任何一家搜索引擎公司都非常重视广告点击率，因为它是一家搜索引擎公司收入运营的重要手段。

广告点击率是互联网广告投放效果的重要指标，它的理论计算值等于广告点击总数与广告展示总数的比。广告投放之后，点击率的影响因素主要包括广告本身方面、搜索上下文方面及搜索用户方面。

（1）广告本身方面。广告本身的类型属性，包括文字广告、图片广告及广告的内容的颜色、字体大小等。

（2）搜索上下文方面。广告位在搜索全文中显示的位置。

（3）搜索用户方面。用户本身的年龄、性别、爱好、地域等。

基于此,在广告投放之前,需要对广告进行广告点击率预测。这种预测并不是针对每个广告都会做一次计算,而是针对某一类广告的点击率预测。广告点击只有两种情况,点击或不点击,以函数方式表示即为:

$$Y = \begin{cases} 1, & \text{点击} \\ 0, & \text{不点击} \end{cases}$$

对于广告的每一个特征,设定为 x_i ,广告中每一个特征对于广告是否被点击的影响力设定 w_i ,因此点击率为:

$$P(y=1)=f(w_i x_i), i=1,2,3,\dots,n$$

不同 f 函数也会导致计算结果不一样,但是通过常规思维,最简单的线性函数为:

$$P(y=1)=w_0+w_1 x_1+w_2 x_2+\dots+w_n x_n$$

由于各自权重相加可能导致概率返回不在区间 $(0,1)$ 之间,因此采用 Sigmoid 函数,公式如下:

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}, t \in (-\infty, +\infty)$$

无论 t 取何值,均可将概率映射到区间 $(0,1)$ 之间,如图 9-39 所示。

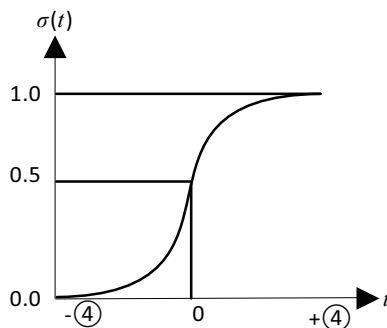


图 9-39 Sigmoid 函数分布情景

通过对 $\sigma(t)$ 的逆变换,则:

$$\theta(t) = \ln \frac{t}{1-t}, \quad t \in (0,1)$$

而 $P(y=1) \in (0,1)$, 因此设定 :

$$\ln \frac{P}{1-P} = w_0 + w_1 x_1 + w_2 x_2 + \dots + w_n x_n$$

$$\begin{cases} P(y=1|x) = \frac{1}{1+e^{-wx}} \\ P(y=0|x) = \frac{1}{1+e^{wx}} \end{cases}$$

其中, $x=(1, x_1, x_2, \dots, x_n)$, $w=(w_0, w_1, w_2, \dots, w_n)$, x 是输入参数, $p(y=1|x)$ 是输出参数, 表示在特征 x 下发生的概率, w 是未知参数。

在给定的公式中, 存在未知参数 w , 因此无法直接对广告点击率进行预估, 需要通过训练, 对参数 w 进行估计, 针对训练样本 :

$$C=(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$$

可以通过最大似然估计的方法, 对于 w 参数进行估计, 假设以固定时间范围作为特征 x , 设定 :

$$\begin{cases} P(y=1|x) = g(x) \\ P(y=0|x) = 1 - g(x) \end{cases}$$

根据似然函数 :

$$\prod_{j=1}^n [g(x_j)]^{y_j} [1 - g(x_j)]^{1-y_j}$$

对数似然函数为 :

$$L(w) = \sum [y_j (w * x_j) - \ln(1 + e^{w * x_j})]$$

因此, 对 $L(w)$ 求最大值, 计算出 w 的估算值即可。

针对历史广告投放数据进行归纳学习, 确定各个特征向量的权重, 权重向量和特征向量属于线性相关。采用逻辑回归, 主要是其模型简单, 处理在线学



习相对比较容易，处理性能也较好，适合用于解决大数据推荐问题。

9.10 搜索引擎评价

没有严格的搜索结果评估标准，搜索技术就难于进步。通过发现缺陷不断改进算法、改进数据结构及系统架构使得搜索引擎技术不断向前发展。作为搜索引擎研发人员，搜索引擎的评价体系一直处于核心地位，搜索引擎架构、算法的优良与评价效果是密不可分的。

9.10.1 搜索评价概述

从笔者个人角度来讲，与其一味地思考评价体系，不如换个角度，搜索引擎的初衷是什么？搜索引擎的初衷是帮助用户以最快的速度找到其最满意的答案。在过去的十年里，搜索引擎一直在准确率、全面性、时效性、多样性等方面努力。

（1）结果的准确率。搜索结果定位准确，减少用户在页面的查找次数。最佳的情况下能够直接命中答案。

（2）结果的全面性。在保障准确率的情况下，能够扩展用户搜索的相关信息宽度，从而减少用户的搜索次数。例如，搜索“康有为”，也能够显示关于“戊戌变法”等相关性信息。

（3）结果的时效性。最新数据是否能够及时展示给用户，尤其是与搜索相关的新闻信息、价格信息等。

（4）检索方式的多样性。多样性语法搜索，支持不同语言、不同文件类型的搜索方式。

（5）搜索时间响应。是指响应一次搜索的时间消耗，时间消耗越短，用户

体验越好。甚至是在用户输入过程中，就能即刻搜索。

其中，时效性、多样性、高速响应现在基本上都已经解决，但是搜索引擎在准确率和全面性上一直不太理想。尤其是大数据时代，为使得结果最佳，结合个性化搜索、网页排序等都是值得再次深入思考的问题。

9.10.2 基于准确率、召回率及 F 值评价

信息检索领域最为广泛且为大众熟知的评价指标是 Precision-Recall（准确率及召回率）方法。基于准确率和召回率的方法更容易被大众理解和接受。

（1）准确率。用于衡量搜索引擎排除不相关数据的能力，通俗理解为返回的结果中有多少是满意的结果。

（2）召回率。用于衡量一个查询搜索到所有相关文档的能力，通俗理解为所有的满意结果，搜索引擎返回了多少。

假设给定一次搜索， A 表示已被定义为满意文档集合。 B 表示本次搜索到的文档集合。则准确率为满意文档和被搜索到的文档的交集数量与满意文档数量的商，公式如下：

$$\text{Precision} = \frac{|A \cap B|}{B}$$

召回率为满意文档和被搜索到的文档的交集数量与搜索到文档数量的商，公式如下所示：

$$\text{Recall} = \frac{|A \cap B|}{A}$$

举例说明，假设已被定义为满意的文档有 80 个，本次搜索一共产生 100 个结果，其中有 75 个是已被定义为满意的文档，则准确率（Precision）为 75/100，等于 0.75，召回率（Recall）为 75/80，等于 0.937 5。

从公式上看，召回率与准确率并无直接关系，但是从实际工程角度来看，尤其是在搜索引擎这种大规模数据量中，两者却又是相互制约的。在搜索过程中，若系统希望返回更多的满意文档集合，则这个时候，也会有很多不满意的文档被搜索出来，从而使准确率受到影响；同理，试图减少不满意文档时，也会使得部分满意文档被排除在外，从而影响到召回率，两者的关系是召回率高时，准确率低，准确率高时，召回率低，趋势基本如图 9-40 所示。

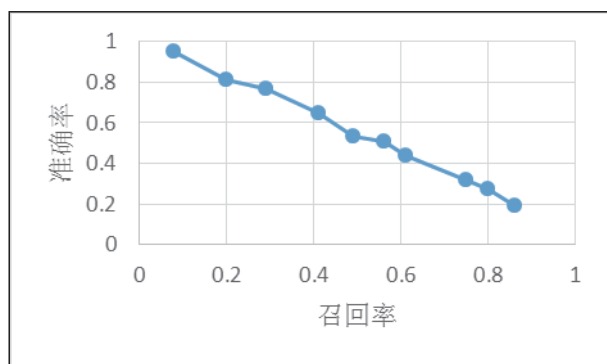


图 9-40 准确率与召回率的粗略关系图

因此，评价的好坏不在于准确率或者召回率单一因素。很难做到两者均处于高位，需要在二者之间寻找平衡，这个平衡后的综合因素，采用 F 度量（F-measure）。F 度量的公式为：

$$F = \frac{1}{\lambda * \frac{1}{\text{Precision}} + (1 - \lambda) * \frac{1}{\text{Recall}}}$$

一般情况下取 $\lambda=0.5$ ，因此可以将公式简化为：

$$F = \text{Precision} * \text{Recall} * \frac{2}{\text{Precision} + \text{Recall}}$$

即 F 为准确率和召回率的调和平均值。调和平均值与通常的几何平均、算术平均不同，调和平均值则强调较小数值的重要性，对小数值比较敏感，因此用于搜索中的准确率和召回率比较合适，因为系统试图尽可能让准确率和召

回率比较高。继续以上例计算 F 度量值，为 $0.75*0.9375*2/(0.75+0.9375)$ ，约为 0.83。

F 值的计算，可以使得准确率和召回率之间得到一定平衡。在搜索引擎中，不能像示例一样仅仅计算一个样本的 F 值，还需要计算更多样本的 F 值。例如，对 1 000 个搜索词进行测试，然后求平均，公式如下：

$$\bar{F} = \frac{1}{n} \sum_{i=1}^n F_i$$

上述内容对计算文档的准确率、召回率及 F 度量值的方式进行分析。在工程中，不仅需要计算准确率、召回率、 F 值，还需要关注文档排序位置的情况。因为对于搜索引擎而言，满意文档的出现并不意味着搜索结果一定满足用户需求，用户不会去一个网页一个网页地浏览，而是从前往后浏览。将文档排序序列纳入评价体系也是必须考虑的因素，因此，还需要利用其他的评价体系去评估排序结果。

9.10.3 归一化折扣累计增益

排序质量衡量指标通过归一化折扣累计增益（Normalized Discounted Cumulative Gain, NDCG）评价，归一化折扣累计增益是一种综合考虑排序和真实序列之间关系的指标。在搜索结果中越相关的网页排序越靠前，分值越高。因为绝大部分用户均是从上向下点击，最相关的放置在最前面，有助于用户减少信息筛选时间。NDCG 公式如下：

$$\text{NDCG}_p = \frac{\text{DCG}_p}{\text{IDCG}_p}$$

其中， DCG_p 表示当前排序结果的折扣累计增益， IDCG_p 表示最佳排序结果下的折扣累计增益。累计增益公式如下：

$$CG_p = \sum_{i=1}^p rel_i$$

rel_i 表示在搜索结果的第 i 位置的评分。折扣累计增益公式如下：

$$DCG_p = rel_i + \sum_{i=2}^p \frac{rel_i}{\log_2 i}$$

NDCG 对于一个已经出现在搜索结果中的页面，不再采用“满意”和“不满意”这样两极的评价方式，而是根据相关度和排序位置采用灵活的“满意、较满意、勉强满意、较不满意、非常不满意”，众多互联网公司也在采取这样的方式。因为这不仅仅关心了数据筛选满意程度，还关心了排序满意程度。搜索结果网页相关性满意度评分如表 9-22 所示。

表 9-22 搜索结果网页相关性满意度评分

满意度	评分
满意	4
较满意	3
勉强满意	2
较不满意	1
非常不满意	0

关于满意度的评分，不一定是 4、3、2、1、0，根据实际情况有所不同，也可以采用指数方式，甚至负分的惩罚机制。

以示例说明 NDCG 评价体系，对于一个搜索词搜索产生的顺序文档为： D_1 、 D_2 、 D_3 、 D_4 、 D_5 、 D_6 。搜索结果中 6 个文档的评分如表 9-23 所示。

表 9-23 搜索结果中 6 个文档的评分

文档	评分 (rel_i)
D_1	4
D_2	2
D_3	3
D_4	0
D_5	1
D_6	1

根据表 9-22，计算增益。

$$CG_6 = \sum_{i=1}^6 rel_i = 4 + 2 + 3 + 0 + 1 + 1 = 11$$

在增益中，即使任意交换两个文档位置，计算结果值依然不变，这样会导致和网页结果的两极化评价方式类似，文档排序序列因素未纳入其中。因此，折扣累计增益计算如表 9-24 所示。

表 9-24 折扣累计增益计算

i	rel_i	$\log_2 i$	$\frac{rel_i}{\log_2 i}$
1	4	0	—
2	2	1	2
3	3	1.585	1.893
4	0	2.0	0
5	1	2.322	0.431
6	1	2.584	0.387

根据表 9-24 显示结果可知，调换任一网页排序序列将会影响到最终评价，因此折扣累计增益结果为：

$$DCG_6 = rel_1 + \sum_{i=2}^6 \frac{rel_i}{\log_2 i} = 4 + 2 + 1.893 + 0 + 0.431 + 0.387 = 8.711$$

要验证本次排序评价，需要与排序最好的情况作对比。对搜索结果文档 D_1 、 D_2 、 D_3 、 D_4 、 D_5 、 D_6 的最佳排序应该是按照分数从高到底排序，结果依次为： D_1 、 D_3 、 D_2 、 D_5 、 D_6 、 D_4 ，对应分数依次为 4、3、2、1、1、0。最佳排序序列情况下的最佳折扣累计增益如表 9-25 所示。

表 9-25 最佳排序序列情况下的最佳折扣累计增益

i	rel_i	$\log_2 i$	$\frac{rel_i}{\log_2 i}$
1	4	0	—
2	3	1	3
3	2	1.585	1.262

续表

i	rel_i	$\log_2 i$	$\frac{rel_i}{\log_2 i}$
4	1	2.0	0.5
5	1	2.322	0.431
6	0	2.584	0

根据表 9-25 显示的结果，计算最佳折扣累计增益：

$$IDCG_6 = rel_1 + \sum_{i=2}^6 \frac{rel_i}{\log_2 i} = 4 + 3 + 1.262 + 0.5 + 0.431 + 0 = 9.193$$

最终归一化累计折扣增益计算结果，得到本次网页排序序列的评分。评分越高，表示效果越好。

$$NDCG_p = \frac{DCG_p}{IDCG_p} = \frac{8.711}{9.193} \approx 0.9476$$

从 $NDCG_p$ 公式中，可以分析出两大特征。一方面在累计过程中表示，越是更多的满意文档，最终结果越好，即保证整体结果质量。另一方面，越是满意的文章越靠前，计算值越高，即保证最佳位置结果质量。

因此， $NDCG_p$ 是一种结合数据筛选和数据排序的评价体系，不仅仅在搜索引擎排序上常常使用，凡是涉及文档排序、数据检索，例如电商、团购等都会使用 $NDCG_p$ 作为排序衡量指数。

值得注意的是，上述对搜索结果的评估方式，需要很大的人力资源，去标注满意结果或者较满意结果，因此大的互联网公司会有“众测”部门，或者将这些工作外包出去。

9.11 本章小结

搜索服务是整个搜索引擎的核心功能之一，良好的用户体验，精准的搜索

结果直接影响用户对于搜索引擎的满意度，良好的用户体验需要支持容错式检索，帮助用户发现搜索过程中的错误，推荐最佳搜索词，还能在最短的时间内返回搜索结果；精准的搜索结果不仅要满足结果的相关性需求，还需要结合用户的个人信息，结合相关性给予用户最佳结果。

广告投放作为搜索引擎密不可分的一部分，同样表现为搜索引擎对用户、对数据的挖掘程度，搜索引擎的检索服务，对技术的要求并不低于搜索引擎后台的数据抓取和分析，尤其是对于广告点击率的预测与搜索引擎的排序同等重要。

通过准确率、召回率、F 值度量及归一化折扣累计增益对搜索服务的评价也使得搜索引擎为用户提供更好的搜索服务体验。

第 10 章 基于用户日志的反馈学习

搜索引擎本身就是一套自我反馈学习系统，不断地在自我反馈中提高搜索准确率、优化搜索排序、个性化贴近用户，而自我反馈部分中，最重要的学习数据就是所有的用户搜索日志记录。

搜索引擎日志是一个非常有价值的数​​据，通过对日志的分析可以给当前用户提供更可靠的关键词推荐、意图分析、规范输入等，对于搜索引擎自我迭代学习其重要性不言而喻。

搜索日志几乎是一个藏宝库，里面蕴藏了太多重要信息，例如，可以发现用户访问时间的分布，完整的一天中，用户使用搜索引擎的高峰在 10 点至 11 点，下午 16 点到 17 点，晚上 20 点到 21 点，最低峰在凌晨 3 点至清晨 7 点，这种周期性数据，可以帮助系统进行升级优化及抗压测试等。

搜索引擎日志挖掘的主要技术有：统计分析方法、数学模型分析与预测、关联规则分析、聚类分析等方法。

10.1 基于用户搜索词语的分析

用户搜索日志的主要目的是根据用户搜索的历史行为分析出用户类型、用户的需求及用户下一步试图想做的事情。

在一次搜索行为完毕之后，搜索引擎为避免该搜索词没有找到很好的结果，在页面尾部一般都会给予数个相关搜索提示，例如搜索“胡歌”，会提示相关搜索“霍建华”“唐嫣”“江疏影”等，这些相关搜索，常常使用关联分析挖掘用户搜索日志给出，以识别用户的查询意图或者用户感兴趣的信息。

10.1.1 发现搜索词的价值

基于搜索日志去发现数据的相关性是一个可行的方法，主要从如下三个方面考虑。

(1) 两个搜索字符串，在众多流量中都相邻被搜索，则两个搜索字符串越相关。此类数据挖掘中最经典的当属于啤酒和尿布的故事，美国的妇女经常在家照顾孩子，少有时间去超市，所以她们经常会告诉丈夫在下班回家的路上为孩子买上一些尿布。然后销售员发现此类丈夫人群在买尿布之后又会顺手买些啤酒。发现啤酒和尿布之间的关系，为商家带来了大量的利润。搜索引擎中的搜索词也存在相应的关系，例如，很多用户在搜索“大数据分析”之后，都搜索了“hadoop”，则说明“hadoop”和“大数据分析”相关。在搜索引擎的相关搜索中，可以充分利用此项关联数据，在用户的搜索词无法实现精准搜索时，为用户提供其他相关搜索词推荐，以使得用户能够找到期望信息。

(2) 发现近义词。一般情况下，比较用户搜索词语集合中两个搜索字符串之间的编辑距离、语义相似度，若两个搜索字符串编辑距离越短、语义相似度越高则两个搜索字符串越相似，为近义词的可能性比较高。发现近义词用于在不同搜索词下能够命中相同的优质结果，一般搜索引擎利用近义词进行搜索相关性推荐，例如，在搜索“最佳男主角”时，推荐搜索“最佳女主角”或“最佳男配角”等。

(3) 高流量的搜索热词优先缓存。从用户不断搜索的搜索词中，可以发现



用户搜索较为频繁的搜索热词，可以估计这些搜索热词最近一段时间的访问热度依然不会太低，因此，优先考虑将高流量的搜索热词进行缓存，使得不再频繁地向后台服务器请求数据，同理也会淘汰部分缓存数据。通过不断地计算每个小时或者每天的搜索热词，可以使得搜索服务尽可能达到较为满意的结果。

10.1.2 发现不明意图下的用户行为

统计表明所有用户的搜索字符串平均长度不超过三个字符，对于单一用户，意图识别难度非常大。再加上用户的不规范输入、语言表达的多样性、甚至方言式表达，这些都使得用户意图识别难上加难。此外，用户的意图也不一定是单一性的，例如用户搜索“大好时光”，用户的可能意图是“大好时光这部电视剧”“出去旅游享受大好时光”等，意图难以揣测。此外，即使是相同的搜索词，在不同的时间搜索，用户的意图也会大受干扰，例如关键词“伪装者”，在多年以前搜索是一个普通角色词，在 2015 年搜索却是一部电视剧，不同的时间给予了不同词语不同的含义。

当然除用户本身与时间导致意图难以揣测之外，还有两点是关于搜索系统本身导致的：

(1) 搜索引擎分词系统中的分词歧义问题。如果分词本身带有歧义，则很难将用户意图揣测清楚，例如，对用户搜索“垃圾漂上海港”则本意是标识垃圾漂到了海港旁边，但是如果分词为“垃圾 / 漂 / 上海港”则很容易将搜索重点落在“上海港”上，而用户搜索本意与“上海港”则没有任何关系。

(2) 互联网中产生的新词不能够及时识别，造成意图特征不明显。典型的例子为电视剧的名称，例如“芈月传”“秦时明月”，如果能够在最短时间内识别到该词为电视剧名称，则可以很好地进行用户意图揣测。

上述问题是搜索引擎本身系统难以解决的复杂性问题，但通过用户搜索日

志，将可以得到很好的解决。

所有的搜索词中，用户难免会在第一页搜索到不满意的结果，并且没能帮助用户解决问题，那么用户下一步会怎么办呢？利用搜索日志分析用户面对无效搜索结果的行为，可以帮助系统改善无效搜索对用户的影响，面对不满意的第一页搜索，大部分用户选择更换关键词、少部分选择继续点击“下一页”，少部分用户也会选择更换搜索引擎尝试。用户的三种行为是在告知搜索引擎研发者需要在两方面着手工作。一方面，完善搜索关键词推荐，尽可能帮助用户提前发现搜索词的不足。另一方面，如果用户点击“下一页”找到了自己满意的结果，则需要将用户搜索的不满意搜索词作为糟糕案例研究。

10.2 基于用户点击日志的分析

用户点击日志主要是用户在使用搜索引擎过程中点击相关链接的日志信息。之所以需要用户点击日志，是因为用户本身不会告诉搜索引擎，当前搜索结果的好或者坏，但是通过用户点击行为，搜索引擎可以进一步分析用户搜索意图及提升网页排序、网页个性化等相关技术。

不仅如此，研究用户在页面的点击分布范围也可以帮助搜索引擎高效利用有效区域，例如广告投放。分析表明，在搜索结果的前 6 条左右的区域是用户频繁点击区域，因此，在此区域适当投放广告，可以增加用户对广告的点击率。

10.2.1 时间与搜索意图的关系

时间是一个不断变化的因素，利用点击日志可以分析出因为时间关系导致的搜索歧义，因为不同搜索词在不同时间段包含的具体含义有一定差异，这种差异性很可能导致搜索结果南辕北辙。

以关键词“小时代”为例，它的歧义性在于可能搜索的是电影或者小说，当用户搜索“小时代”时，结果集中在电影还是小说上成为搜索意图分析的重点。通过在最近一个月的搜索日志中进行挖掘，发现搜索“小时代”的点击日志存在流量变化。将“小时代”点击日志中被点击的文档类别进行分析，大致分为两类：“电影”相关和“小说”相关，并将点击次数按照流量统计出，如图 10-1 所示。

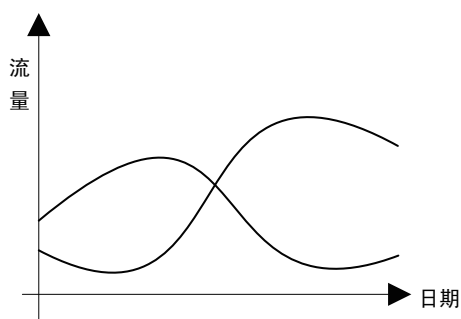


图 10-1 “小时代”关于“电影”相关与“小说”相关流量

图 10-1 中，虚线为搜索“小时代”后点击“小说”相关网页流量曲线图，随着时间的发展，流量逐渐减少。对于实线，是搜索“小时代”后点击“电影”相关网页浏览曲线图，两者最近一周流量比约为 4:1，若当前时刻用户搜索“小时代”，则用户期望获得小时代电影相关信息的可能性约占 80%，期望小说相关的可能性约占 20%。

搜索引擎的日志分析系统不间断地进行各类搜索热词可能的倾向性结果分析。相关示例还包括“芈月传”“琅琊榜”等。

10.2.2 地理位置与搜索意图的关系

地理位置作为搜索引擎非用户输入的基本数据之一，它对搜索引擎的个性化起到非常重要的作用。因为地理位置不同，即使是搜索相同的关键词，用户

的搜索意图和期望也不尽相同。以关键词“交通大学”为例，它的歧义点在于可以被理解为“北京交通大学”“上海交通大学”“西安交通大学”“重庆交通大学”“大连交通大学”等，虽然都是交通大学，但都是针对“交通大学”这个关键词。

面对不同地域的搜索，可以通过点击意图也不一致进行分析，依然以“交通大学”为例，在某一时间范围内在不同地区的搜索与点击如表 10-1 所示。

表 10-1 关于搜索“交通大学”的地理位置与意图关系

搜索来自地区	点击“北京交通大学”次数	点击“西安交通大学”次数	点击“上海交通大学”次数
京津冀地区	70 万	1 万	30 万
江浙沪地区	1 万	1 万	100 万
大西北地区	1 万	65 万	35 万

通过表 10-1 可以发现，各个地区都有对“交通大学”是“上海交通大学”的简称的认可，但是由于各地也有属于自己的交通大学，例如从点击次数上分析，大西北地区有约 65% 的用户认为“交通大学”指的是“西安交通大学”，京津冀地区有约 70% 的用户认为“交通大学”指的是“北京交通大学”，因此在搜索引擎的个性化中，也需要根据用户所在位置确定最佳的排序，目前主流的搜索引擎都支持基于地理位置的个性化排序。例如，重庆的搜索引擎用户在百度中搜索“交通大学”的第一条结果为“重庆交通大学”，如图 10-2 所示。



图 10-2 重庆地域用户在百度中输入“交通大学”搜索结果第一条

同理，在西安搜索“交通大学”后，在搜索结果中点击“西安交通大学”相关页面的概率高于“北京交通大学”。因此，根据地理位置区域挖掘搜索意图也是非常重要的一环，类似还有“工商管理局”“理工大学”“中山路”等，都属于挖掘本地日志进行本地化搜索。

但这些信息并非一成不变，需要对用户的点击日志进行周期性分析，比如到了一定时间，大家都认为“交通大学”这个词就是指上海交通大学，那么搜索结果中，在北京、西安、重庆搜索“交通大学”的第一条结果均是“上海交通大学”。

10.2.3 点击日志与同义词

对于两个不同的搜索字符串，两个搜索结果中被用户点击的链接相同的越多，则两个字符串越相关。例如，搜索词 A 和搜索词 B，结果页中存在 K 条相同链接，且这 K 条链接中有多条均被用户点击，则说明，搜索词 A、B 相关。例如，搜索“中国科学技术大学”与“中国科技大学”的结果如表 10-2 所示。

表 10-2 搜索“中国科学技术大学”与“中国科技大学”的结果

搜索词	搜索结果中热门点击文档	点击文档对应链接
中国科技大学	1. 该校官方首页 2. 该校研究生招生信息网站 3. 该校就业信息网站	1. www.ustc.edu.cn 2. yz.ustc.edu.cn 3. www.job.ustc.edu.cn
中国科学技术大学	1. 该校官方首页 2. 该校本科生招生信息网站 3. 该校就业信息网站	1. www.ustc.edu.cn 2. zsb.ustc.edu.cn 3. www.job.ustc.edu.cn

由表 10-2 可知，在搜索“中国科学技术大学”与“中国科技大学”的热门点击结果中，它们之间存在一定的交集，这种交集信息即为两者相关性的体现，并且两者在热门点击中的差异性并不明显，不同之处在于热门点击的第二条记录“中国科学技术大学”对应的是“本科生招生信息网站”，而“中国科技大学”对应的是“研究生招生信息网站”。如果将此项的差异性进一步以用户群分析，

会发现只是不同群体对“中国科学技术大学”的称呼方式不一样而已，热门点击文档不一致也是因为各类用户群体对该校的关注点不一样，有的关注本科生教育，有的则关注研究生教育。

按照相同的方法，可以发现很多的同义词，例如“哈尔滨工业大学”与“哈工大”、“上海交通大学”与“上交大”等。

同义词的发现可以帮助搜索引擎聚焦更加精准的搜索结果。例如，搜索“哈工大”如果不能得到较好的结果，而搜索“哈尔滨工业大学”的效果较好，则考虑将“哈工大”转化为“哈尔滨工业大学”进行搜索。

10.2.4 点击日志与词语权重

在网页进行相关性排序时（参见 9.1.4 节），会根据词语的权重及词语在文档中的权重综合考虑排序序列，词语的权重源自离线计算的样本 TF-IDF 值，但是存在这样的情况，样本的不同可能导致词语对应的 TF-IDF 值不同，因此需要通过用户的日志进行校验优化。

例如，针对“大数据分布式计算”的分词结果“大数据”和“分布式计算”，初步获得的 TF-IDF 值为别为 0.23、0.23，但是对“大数据分布式计算”返回的结果中，综合大量用户的点击情况如表 10-3 所示。

表 10-3 “大数据分布式计算”搜索结果中点击量分析

结果类别	点击量
“大数据”与“分布式计算”共同相关结果	16 万
仅“大数据”相关结果	3 万
仅“分布式计算”相关结果	1 万

从表 10-3 中可以直观地看出，用户的最佳期望是“大数据”与“分布式计算”的共同结果，然而在不能满足最佳期望时，用户点击的“大数据”相关结果比“分布式计算”相关结果高。从搜索引擎角度分析，在“大数据分布式计算”这个

搜索词中词语“大数据”应当比“分布式计算”更加重要。可以通过图 10-3 所示的计算方法，更改“大数据分布式计算”中“大数据”和“分布式计算”的权重比例，其中，0.85 是阻尼系数，0.15 是 1 与阻尼系数的差。

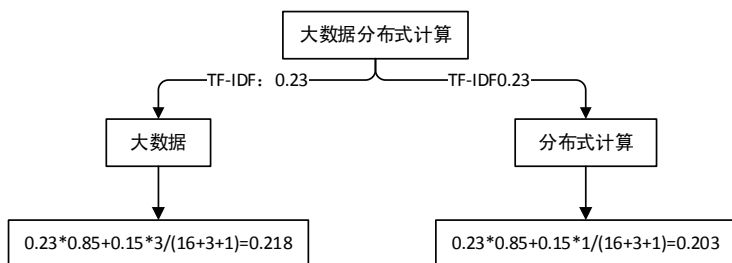


图 10-3 词语“大数据分布式计算”中根据点击重新调整权重

上述过程中并不是修改了“大数据”和“分布式计算”的 TF-IDF 值，而是修改了当两个词语同时出现时的权重比例，例如计算出的权重分别为 0.218、0.203，意味着在词语“大数据分布式计算”中“大数据”的权重应当是“分布式计算”的 1.07 倍，如果“分布式计算”的 TF-IDF 值为 0.23，则计算时“大数据”的临时 TF-IDF 值为 0.247。

类似例子还包括用户输入“iPhone 手机”，通过分析可以发现实质的关注点在“iPhone”并不在于“手机”，如果这个时候尽可能展示“iPhone”相关的搜索结果将会有助于搜索体验的提升，所以针对词语的权值进行分析，实质也是对于网页排序的局部优化。

10.2.5 点击日志与新词分类

识别新词在真实的语义环境中的分类。一般情况下，可通过对分类语料库中的词语进行多分类标注，使得可以知道词语最可能的一些分类情况。但是对于刚刚出现的新词，则需要在尽可能短的时间内确定该词可能的分类，以使得下一次能够精确对搜索词进行分类，然后定位搜索领域。

一般情况下，在对用户的搜索词进行分类时，基本上都能够较好地区分出所属分类，如图 10-4 所示。

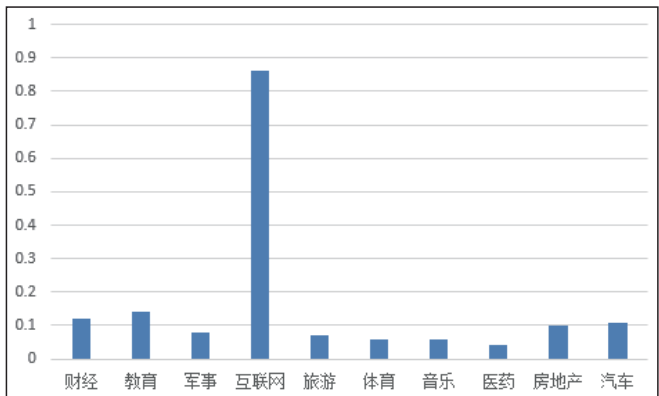


图 10-4 某词语具备特定分类的多分类权值分布

对于新词进行同样的分词过程中，可能会发现新词的分类结果不仅在各分类上的值不高，且不能区分出一个代表性的分类，如图 10-5 所示。

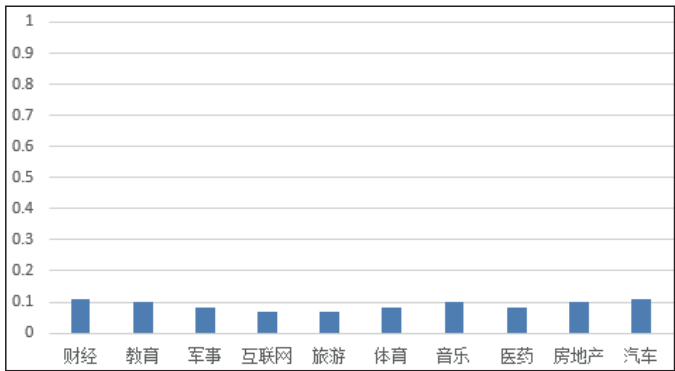


图 10-5 某词语不具备特定分类的多分类权值分布

面对图 10-5 中的情况，则需要利用搜索点击日志去发现新词与分类之间的相关性。用户不会直接告诉搜索引擎当前输入的词属于什么分类，但是大量用户的点击行为从侧面反映出词语的分类，某未知分类的新词 A 对应点击的文档分类及次数如表 10-4 所示。

表 10-4 某未知分类的新词 A 对应点击的文档分类及点击次数

点击文档的分类	点击次数
财经	10 万
教育	2 万
音乐	1.5 万
互联网	1 万

如表 10-4 所示，搜索新词 A 的搜索结果的热门点击文档中主要包括四类，其中 10 万次对财经类相关文档点击、2 万次对教育类相关文档点击，1.5 万次对音乐类相关文档点击、1 万次对互联网相关文档点击，则说明新词 A 的分类很可能为财经类，如果以数值化表示，则可以利用该分类下的点击次数与总点击次数相比。上述过程实质也是利用众包的思想进行的新词分类标注。

10.2.6 点击日志与知识图谱

知识图谱在大多数情况下是通过对百科网站数据进行结构化和非结构化分析得到的结果，但是用户的点击日志却也能够为知识图谱提供可靠的数据分析源。搜索引擎不会将互联网中的所有网页进行实体抽取，但是会选择性抽取部分网页进行实体和实体关系抽取。这里的选择性指用户点击文档的标题。例如，对于搜索词“马云老婆”，在搜索引擎的知识图谱中并未找到，但是通过网页搜索得到的部分网页标题，如下所示：

马云老婆张瑛简介及家庭背景

马云老婆张瑛和儿子照片资料大曝光

马云老婆张瑛谈幸福，马云妻子、女儿、儿子照片大曝光

马云妻子张瑛_马云老婆是谁？张瑛背景资料简介及照片曝光

马云漂亮老婆张瑛简介及珍贵照片曝光

...

虽然搜索引擎无法在知识图谱中获得马云老婆是张瑛的信息，但是用户可以从网页中直接知道马云的老婆是张瑛。同理，搜索引擎也可以按照同用户一样的方式，重新获知实体“张瑛”，获取的过程可以通过用户在输入“马云老婆”后的热门点击日志分析，如表 10-5 所示。

表 10-5 搜索“马云老婆”之后用户热门点击文档的标题及次数

文档编号	点击文档的标题	点击次数
1	马云老婆张瑛简介及家庭背景	10 万
2	马云老婆张瑛和儿子照片资料大曝光	2 万
3	马云老婆张瑛谈幸福马云妻子、女儿、儿子照片大曝光	1.5 万
4	马云妻子张瑛_马云老婆是谁? 张瑛背景资料简介及照片曝光	1 万
5	马云漂亮老婆张瑛简介及珍贵照片曝光	1 万

将表 10-5 中的点击热门文档的标题进行实体抽取，可以发现它们的共同特征，均可以提取出两个实体“马云”“张瑛”，且两个实体之间的关系是“老婆”，“张瑛”是关系“老婆”的接受实体。

不是所有的网页标题都可以进行实体和实体关系抽取，只有搜索词对应的热门网页标题才可以，而且需要在这些热门的网页标题中，大部分都能抽取这些实体及实体关系。之所以会以用户点击及热门网页标题进行抽取是因为这种方式不仅可以过滤掉部分价值较低的网页还保证了实体抽取的时效性。时效性是指在当前时间范围内，用户都会觉得该项信息有效，例如，目前阿里巴巴的首席执行官是张勇，而几年以前是马云，但是两者并存的网页都会被搜索引擎找到，通过用户的热门点击，则可以逐步确定目前的阿里巴巴首席执行官是张勇而不是马云。

10.2.7 点击日志与网页重排序

如图 10-6 所示，对于搜索“北京市政府”产生的两个排序不定的文档，通过用户点击确定文档对用户的价值，在不断的累计过程中，文档价值越高排序越靠前。

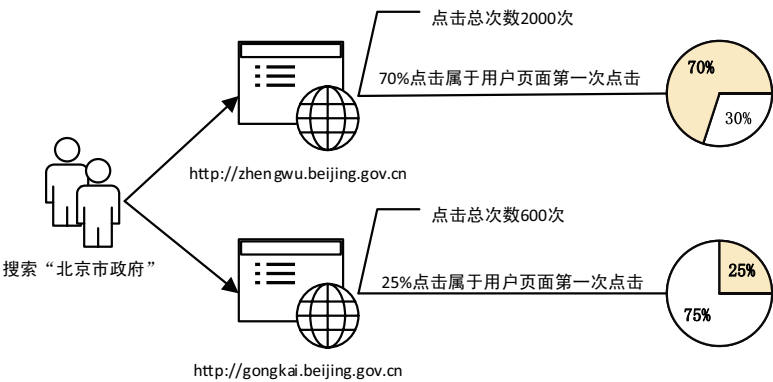


图 10-6 搜索“北京市政府”结果页中的用户点击情况

基于图 10-6 所示，虽然累计链接点击次数不相同，但是一般越是用户直观感觉比较靠谱的网页，点击次数越多。此外，在页面内链接中，第一次点击概率高的，也是属于用户觉得最靠谱的页面，排序也会相对靠前。

用户每点击一个链接，则是对这个链接文档与搜索词之间的一个评分，被第一个点击的链接评分最好。这对于跳转搜索排序非常重要。搜索结果页的默认排序与评分如表 10-6 所示。

表 10-6 搜索结果页的默认排序与评分

链接	默认排序	默认评分
url 1	1	0.98
url 2	2	0.97
url 3	3	0.89

但是，点击分布并不是完全按照默认顺序，用户对链接点击情况统计与评分如表 10-7 所示。

表 10-7 用户对链接点击情况统计与评分

链接	第一个被点击次数	用户评分指数
url 1	489	$489/(489+521+61) \approx 0.456$
url 2	521	$521/(489+521+61) \approx 0.486$
url 3	61	$61/(489+521+61) \approx 0.057$

通过表 10-7 可知, 用户对于 url 2 和 url 1 的排序位置存在犹豫。两者在第一次被点击次数中比较相近。因此, 针对用户点击给予的评分与原始网页排序的评分进行综合, 从而进行重排序。网页原始评分与用户评分的结合重排序如表 10-8 所示。

表 10-8 网页原始评分与用户评分的结合重排序

链接	累计得分	重新排序
url 1	$0.98+0.456=1.436$	2
url 2	$0.97+0.486=1.456$	1
url 3	$0.89+0.057=0.947$	3

上述通过用户点击链接评分的方式, 以用户人工标记的方式确定谁是第一位或第二位, 对于本身默认评分中细微差距的搜索结果, 系统难以分出绝对的第一位置和 second 位置, 通过用户的点击效果, 逐步拉开差距, 并给予新的评分, 重新排序。这也是搜索引擎越多人使用, 效果越好的原因之一。

10.2.8 点击日志与网页评价

评价某个页面是否符合用户预期, 用户点击仅仅说明看起来很像, 是否真实地对用户具有价值, 还需要通过被跳转页面被用户浏览的驻留时间, 例如用户搜索“新加坡自助游”, 点击跳转到 <http://a.com/xinjiapo> 页面, 而去统计用户在 <http://a.com/xinjiapo> 页面的行为, 尤其是驻留时间, 驻留的时间越长, 越说明网页结果可靠性越好。针对这一点, 搜索引擎利用了用户点击的时间差进行分析。主要分为如下几种情况:

(1) 当大量用户进行一次相同搜索之后, 对返回结果页中的链接 A 点击之后, 绝大部分用户也再没有点击搜索结果页的其他链接, 则说明链接 A 对应的网页信息符合用户的预期较高。

(2) 如果用户点击了链接 A 之后, 很多用户在较短时间内, 返回了搜索页面, 重新点击了链接 B, 且不再点击搜索结果页面内的其他链接, 则说明 B 链



接对应的页面相对于链接 A 对应的页面有一定的可靠性。

(3) 如果在页面内点击了链接 A 之后还不断地点击了页面内的更多链接, 则说明, 用户在该搜索词下并未得到较好的期望结果, 可能与用户的搜索词有关, 也有可能与搜索引擎的排序有关。

(4) 如果用户首先点击了排序靠后的链接 B, 然后重新点击了排在前面的链接 A, 则说明文档排序较为合理, 导致用户优先点击链接 B 的原因可能是文字相关性较高, 链接 A 可能个性化程度较高。

上述四种情况作为基本的链接分析情形, 都是在海量的搜索日志中进行分析的, 而且在工程应用中还会更加复杂。

为了达到更好的页面驻留时间分析, 各大搜索引擎厂家在搜索引擎发展的早期, 都大力投入浏览器研发, 因为浏览器不仅仅是网络访问工具, 还成为各大厂家提升各自搜索引擎效果的利器。浏览器可以辅助搜索引擎进行行为分析, 最近几年浏览器战火烧到了移动端, 搜索引擎与浏览器的关系是非常值得去深入思考的问题。

10.3 基于用户的特征分析

用户特征是用户的独特属性, 通过用户信息不断挖掘用户数据中的价值。被用于用户特征分析的数据源主要来源于如下三方面。

(1) 注册信息。这部分信息是用户在搜索引擎网站或搜索引擎联盟网站注册过程中自己填写的信息。

(2) 用户在使用搜索引擎中被记录下的每个行为, 包括搜索词、点击链接、翻页行为等。

上述两部分用户信息，看似杂乱无章没有必然关系，但是通过“用户跟踪”技术，可以有效地将两者信息串联起来，得到较好的用户信息数据集。这些数据集再通过用户群体行为特征分析和用户个体行为特征分析，最终能够对用户的特征有较好的描述。

10.3.1 用户跟踪

越是一个持久使用的用户，对搜索引擎的后期分析效果越好，也非常利于广告推荐。再加之，仅仅通过用户每天的搜索，很难分析出用户的所有爱好、学历等各种信息，搜索引擎无法完整地跟踪用户行为路径。因此，搜索引擎常常以 Cookie 的方式，记录用户某些行为，这些行为不仅包括搜索词，还包括在别的网站上的行为。

搜索引擎一般会在其官网标识某用户的固定 Cookie 内容，例如 UID=BCZ8B（实际更长区分度）之类。在不同域名中，这个 UID 也属于该搜索引擎旗下的网站 A，通过嵌套搜索引擎官方外链共享 UID。当然在其广告联盟会员中的其他网站中也会共享到 UID，大致如图 10-7 所示。

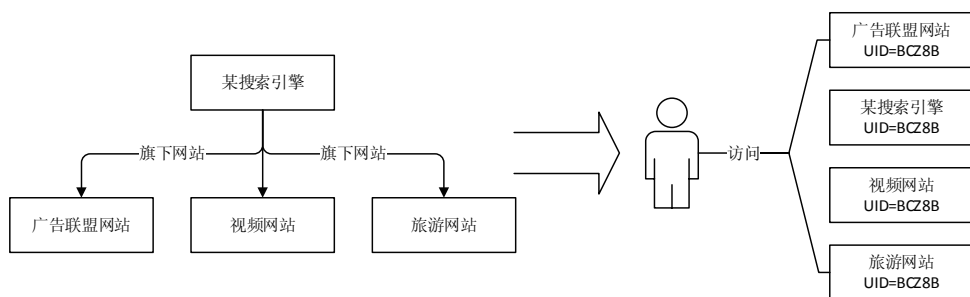


图 10-7 利用“用户跟踪”技术跟踪用户行为

浏览器也可以设置“禁止跟踪”（Do Not Track, DNT）标识，“禁止跟踪”是 W3C（国际互联网联盟）提出的网络隐私保护标准，目前微软、谷歌等公司的浏览器产品相继支持该标准。DNT 实质上是 HTTP 一个头字段，带有 DNT

字段的 HTTP 头信息将会新增“DNT:1”标识。

采用用户跟踪的最初出发点是为搜索引擎能够进行有效的数据分析，但是搜索引擎还能够将这些精准的用户数据与广告结合，从而进行广告更加精准的投放以增加广告点击率，获得较高的广告收益。从搜索引擎本身的技术角度或者搜索引擎的商业价值考虑，进行用户跟踪有很重要的意义。

10.3.2 用户群体特征

要对用户的群体特征进行分析，可以根据用户的行为相似性对用户进行聚类分析。一般认为相似用户的搜索行为可能类似，将此类用户有效地聚集在一起，可以有效地进行搜索行为预测及提升用户搜索体验。

要对用户进行聚类，则需要从用户的搜索历史行为记录开始分析，例如存在四个用户 A、B、C、D，他们的搜索行为如表 10-9 所示。

表 10-9 用户、搜索词与点击目标类型关系

用户	搜索词	点击目标类型
用户 A	“大话西游”	相关电影
用户 B	“大话西游”	相关游戏
用户 C	“生化危机”	相关电影
用户 D	“生化危机”	相关游戏

基于表 10-9 来看可以分析出用户 A、C 属于电影迷类型，用户 B、D 属于游戏迷类型。现在存在另一用户 E 搜索了“波斯王子时之刃”，那么如何预测用户 E 对于搜索结果更倾向于电影还是游戏？

分析用户的意图倾向，如果与用户 E 相似的用户大多数都倾向于电影迷，则用户 E 搜索结果倾向很可能是电影。相似用户分析中的聚类方法，是将零散感觉无相关交集的用户聚集为一个集体，如图 10-8 所示，认为在同一聚簇中的用户具有相似行为或特征。

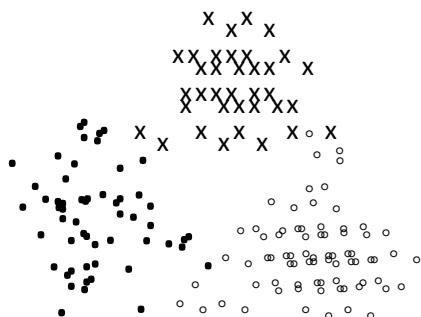


图 10-8 用户聚类大致效果

而聚类的过程是根据用户的历史搜索记录，去发现搜索用户之间的相互关系。例如，用户 A 曾经有很多关于“旅游”的搜索记录，用户 E 也曾经搜索过很多关于“旅游”的搜索词，则说明用户 A 和用户 E 存在相似爱好，用户 B 曾经搜索过很多关于“越野车”的搜索词，用户 E 也曾经搜索过关于“家用轿车”的搜索词，则说明用户 B 和用户 E 也具备共性。而用户 A 和用户 C 都是电影迷，因此，用户 E 是电影迷的可能性比较高，用户 E 搜索“波斯王子时之刃”点击意向更多倾向与电影相关，如图 10-9 所示。

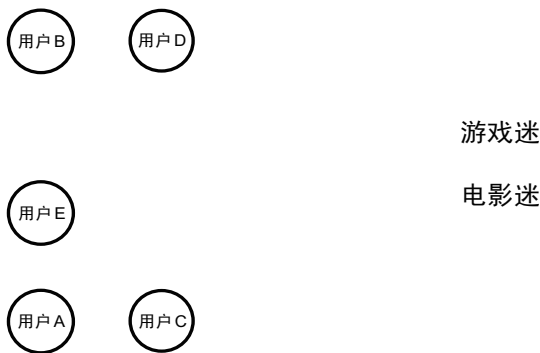


图 10-9 用户之间根据特征划分聚类簇

用户之间行为越相似，距离越近，越符合“人以类聚，物以群分”的思想。将用户以群体的方式进行分析，还可以依据用户搜索历史记录对用户进行独立个体特征分析。

10.3.3 用户个体特征

针对 10.3.2 中的例子，采用基于群体相似性进行分析可以预测用户可能的结果倾向，但是同样可以通过对其用户个体进行研究得到结果倾向。

例如，用户 E 最近一天搜索历史记录：“肖申克的救赎”“阿甘正传”“马来西亚旅游攻略”“辛德勒的名单”“北京自助游”“开心消消乐”等，则需要分析用户搜索“生化危机”的意图是相关电影还是相关游戏。对用户 E 历史记录进行统计分析发现，有超过 60% 的搜索与电影相关，30% 的搜索与旅游相关，少于 10% 的搜索与游戏相关，则显然搜索“生化危机”与电影相关的可能性较高，如图 10-10 所示。

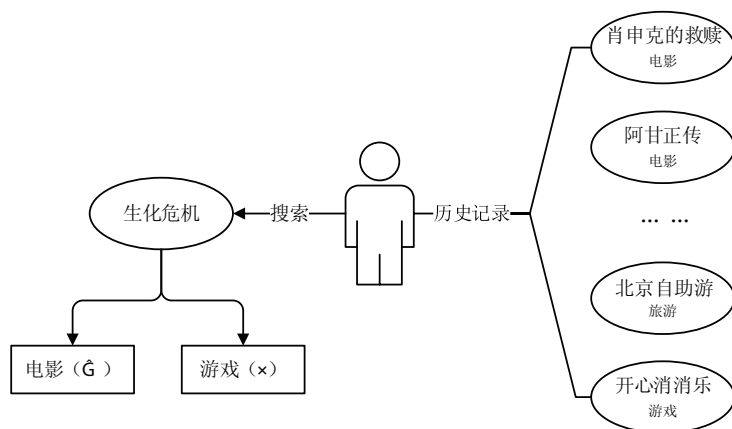


图 10-10 用户搜索“生化危机”结果倾向性

将用户视为独立个体研究时，不仅是研究搜索历史记录的分类问题，还包括输入预测。在智能提示中，如果按照搜索热度，当用户输入“耳”字之后，在下拉提示框中会按照如下顺序输出“耳机”“耳朵”“耳麦”“耳机线”“耳塞”“耳鸣”等提示词，这样的顺序是基于用户群特征分析的结果，即根据广大网民在输入“耳”后选择词语的概率进行提示。

但是如果某用户最近的搜索历史记录是“感冒了怎么办”“耳朵不舒服”

“耳朵保暖”等，那么当此用户在搜索框中输入“耳”字时，依然按照热度提示，则不一定满足此用户的需求，最佳提示应当是“耳朵”“耳鸣”等依次提示。个体特征分析可以使得与用户息息相关的应当优先提示，个体特征分析是针对搜索中个性化搜索的基本方法之一。

10.4 本章小结

基于用户日志的反馈学习中，依然还存在更多的数据价值可以被发现。根据需要会不断挖掘其中的价值。对于任何一家搜索引擎公司而言，日志都是其核心数据之一。日志不仅是挖掘用户的突破口，也是发现搜索引擎不足的窗口，只有从各个角度去挖掘数据，才能去真实地发现用户的想法，搜索引擎从了解用户、认识用户，最终成为用户知心、贴心伴侣。日志是搭建在用户和搜索引擎之间的桥梁。搜索引擎也通过日志分析不断反馈学习，使得自身在技术架构、数据结构、算法分析等方面不断得到完善，为用户的搜索体验带来巨大提升。

鸣谢

本书的内容几乎全部来自我的工作经验，我要感谢本人曾经的导师于炯教授、叶勇教授，以及在微软和百度工作时的老大王明雨、何绍建、黄诚，正是你们曾经对我的悉心指导，才能使得我有能力和勇气写出这本书，无论你们在哪里，凡平永远心存感激。当然还有那些曾经一起学习的同学、共事的朋友，你们曾经给予我很多无私的帮助，使得自己在和你们相处的过程中快速成长，感谢一路上有你们的陪伴，正是有你们，沿途的风景才格外美丽。

感谢英国的 Peter Boden 先生，虽然我们素未谋面，但是你却一直支持我们团队去实现自己的梦想，并连续两年无偿支持我们在 Github 开源社区的项目：Iveely 开源搜索引擎。如果没有你的支持，也许我们不能正常完成 Iveely 开源搜索引擎的研发任务，更不可能去挑战搜索引擎与人工智能的新高度，对你的感激之情溢于言表。

感谢我的创业伙伴魏琪，当我提出我们一起研究人工智能技术时，你毫不犹豫地选择和我一起艰苦创业。无论处在创业的何种困难期，你总是为团队鼓气，你踏实和认真的工作态度让我钦佩不已，感谢你为本书的技术细节提出了真知灼见的修改意见，还有华米神童李赛及杜彬，都对本书提出了改进意见，对此也表示深深的感谢。

感谢北京源智天下公司的吉编辑及出版社的各位编辑，对本书的内容都给予了极大的帮助，对本书的出版也付出了辛苦汗水。

衷心感谢我的家人，感谢你们在过去一年中对我的理解和支持，为我营造了一个良好的写作环境，并鼓励我坚持认真写作，使得本书能够顺利完成。

本书编写过程中还得到了很多朋友的支持和帮助，限于篇幅，虽然不能一一对你们表示感谢，但是我对你们一样表示感激。

最后，感谢这个时代，给予每个有理想的人，赋予实现人生价值的机会！

反侵权盗版声明

电子工业出版社依法对本作品享有专有出版权。任何未经权利人书面许可，复制、销售或通过信息网络传播本作品的行为；歪曲、篡改、剽窃本作品的行为，均违反《中华人民共和国著作权法》，其行为人应承担相应的民事责任和行政责任，构成犯罪的，将被依法追究刑事责任。

为了维护市场秩序，保护权利人的合法权益，我社将依法查处和打击侵权盗版的单位和个人。欢迎社会各界人士积极举报侵权盗版行为，本社将奖励举报有功人员，并保证举报人的信息不被泄露。

举报电话：(010) 88254396; (010) 88258888

传 真：(010) 88254397

E - m a i l: dbqq@phei.com.cn

通信地址：北京市万寿路 173 信箱

电子工业出版社总编办公室

邮 编：100036